KAUNO TECHNOLOGIJOS UNIVERSITETAS
INFORMATIKOS FAKULTETAS
INFORMACIJOS SISTEMŲ KATEDRA

Darius Žilinskas

Dinamiškų internetinių sistemų kūrimo metodika

Magistro darbas

Darbo vadovas
prof. dr. Lina Nemuraitė

Kaunas, 2007

Turinys

Terminų ir santrumpų sąrašas **2**
1. ĮVADAS **5**
2. DINAMIŠKŲ SISTEMŲ (PUSLAPIŲ) POREIKIS IR KŪRIMO METODAI **9**
2.1. ANTROS KARTOS INTERNETAS (WEB 2.0) 9
2.2. AJAX (ASINCHRONINIS JAVASCRIPT IR XML) 14
2.2.1 Architektūra, veikimo principai *14*
2.2.2 Naršyklės modelis Ajax pagrindu *19*
2.2.3 Ajax karkasai *20*
2.2.4 Ajax privalumai ir trūkumai *22*
2.3. AJAX IR TAPESTRY KARKASAS 24
2.3.1 Tapestry karkasas *24*
2.3.2 Tapestry karkaso pasirinkimo pagrindimas *25*
2.4. PAŽINTIS SU JSON (JAVASCRIPT OBJECT NOTATION) (ALTERNATYVA XML FORMATUI) 26
3. DINAMIŠKOS SISTEMOS BAZINIO AJAX MODELIO APRAŠYMAS **27**
3.1. PANAUDOJIMO ATVEJAI 27
3.2. DINAMIŠKO PUSLAPIO ARCHITEKTŪRA 28
3.3. KLASIŲ DIAGRAMOS 32
3.3.1 Statinis vaizdas *32*
3.3.2 Dinaminis vaizdas *47*
4. SISTEMOS PROGRAMINĖ REALIZACIJA IR DINAMIŠKŲ PUSLAPIŲ SU AJAX TYRIMAS **53**
4.1 EKSPERIMENTINĖS SISTEMOS APRAŠYMAS 53
4.1.1 Sistemos funkcijos *53*
4.1.2 Dinamiškų sistemų (puslapių) realizacija su Ajax ir Tapestry *55*
4.2 DINAMIŠKŲ PUSLAPIŲ SAVYBIŲ TYRIMAS IR ĮVERTINIMAS 59
4.3. ĮVERTINIMŲ SUVESTINĖ IR IŠVADOS 63
5. DARBO APIBENDRINIMAS **64**
6. IŠVADOS **65**
Literatūra **67**
Priedai **70**
PRIEDAS 1. POKALBIŲ SKAIČIAVIMO GRAFIKAI 70
PRIEDAS 2. POKALBIŲ SISTEMOS VIDINIŲ VEIKSMŲ SEKIMO (LOGS) IŠTRAUKA 71
PRIEDAS 3. STRAIPSNIS, SKAITYTAS „INFORMACINĖ VISUOMENĖ IR UNIVERSITETINĖS STUDIJOS (IVUS‘07) „ KONFERENCIJOJE. 72

Terminų ir santrumpų sąrašas

Terminas	Aprašymas
MVC	Model-View-Controller – architektūra, kur aplikacija padalinama į 3 skirtingas, atskiras dalis: vaizdą, modelį bei valdiklį.
AJAX	Asynchronous JavaScript+CSS+DOM+XMLHttpRequest technologijų rinkinys.
GUI	(graphical user interface) Grafinis vartotojo interfeisas.
DAO	(data access object) dizaino šablonas, atskiriantis žemo lygio duomenų prieigą, nuo aukšto lygio biznio logikos.
JSP	(java server page) normalus HTML puslapis su Java kodu. JSP kompiliuotojas JSP paverčia į Servletą.
Servlet	Serverio pusės Java programa, kuri suteikia papildomų savybių serveriui. Iš Servleto yra generuojamas HTML kodas.
HTML	(HyperText Markup Language). Web puslapio aprašymo kalba.
HTTP	(HyperText Transfer Protocol). Standartinis web puslapių perdavimo protokolas.
URL	(Uniform Resource Locator). Internetinis web resursų adresas, suprantamas ir interpretuojamas web naršyklių.
JSF	(Java Server Faces) Struts karkaso tęsinys.
CSS	(Cascading Style Sheets). HTML dokumenų formavimo, stilizavimo kalba.
XML	(eXtensible Markup Language). Universali kalba.
DOM	(Document Object Model). Medžio tipo struktūra, laikanti visus HTML objektus.Naudojama Javascript ir kitų skriptavimo kalbų, kad dinamiškai formuoti ir keisti HTML puslapius.
API	(Aplication program interface) Aibė šablonų bibliotekose, kurios išplečia kalbos funkcionalumą.
EJB	(Java Enterprize beans)
JDBC	(Java database connectivity). Sun Microsystems standartas apibrėžiantis aplikacijų priėjimą prie duomenų bazių.
OGNL	**(**Object Graph Navigation Language), išraiškų kalba Java objektų savybės nuskaityti bei nustatyti.
POJO	(Plain Old Java Object) java objektas
XLS	(Xml stylesheets), xml aprašomas puslaio stilius formavimnas ir panašiai.
AOP	(Aspect-Oriented Programming) programavimo technika.

Folksonomy	Tai vartotojo sukurta sistematika, naudojama internetinių puslapių, nuotraukų, nuorodų ir kito žiniatinklio turinio gavimui ir skirtymui į kategorijas
Blog	Asmeninis dienoraštis žiniatinklyje.
Wiki	Interneto svetainė, leidžianti patiems vartotojams bendrai redaguoti ir keisti jos turinį.

Methodology of creating dynamic internet systems
Summary:

This work is concentrated on creation of dynamic web systems. Web 2.0 brings new requirements to internet based systems (web pages). Users needs to work with large amounts of data (information) using simple web browser, are increasing very fast. So there is a problem when we need to represent this data on web page. We need dynamic systems (pages) to do it fast ant easy to operate. Dynamic system (page) is internet based system (web page), which most functions are using Ajax (asynchronous communication with server). A basic architecture model for dynamic systems (pages) is created. It is based on Spiar architecture model. Spiar model was adapted to our dynamic systems (pages). It is described in detail with static and dynamic UML diagrams. At last the real experimental system for dynamic systems model is created and analyzed.

Document is divided in three main parts. The first one is about Web 2.0, its used technologies, need of Ajax and etc. The second part describes the basic architecture model for dynamic systems (pages). And the third last large part is about experimental representation of created model.

At last findings of this master work are:

1) Web 2.0 has brought us demands of dynamic internet based systems (typically with Ajax).
2) Created dynamic systems (pages) model showed that performance of experimental system user interface functions is increased by about web page opening time, where the used function is.
3) Experiments results showed that the model working characteristics are good but should be improved to start use the model in real mass usage systems.

1. ĮVADAS

Beveik kiekvienas šiuolaikinis žmogus nebeįsivaizduoja savo gyvenimo be interneto. Internetas tampa nebeatskiriama gyvenimo dalimi. Taip nutiko todėl, kad kaip ir technologijos tobulėja – tobulėja ir interneto teikiamų paslaugų spektras bei kokybė. Atsirado tokios sąvokos kaip Web 2.0 (antros kartos internetas), dinamiški puslapiai (sistemos), Ajax ir kiti. Interneto vartotojai nori iš interneto gauti vis daugiau ir daugiau paslaugų. To pasekoje kai kurios darbastalio taikomosios programos yra perkeliamos į internetą ir internetas tampa lyg platforma. Sparčiai didėja duomenų kiekiai, kuriuos reikia pateikti vartotojui ir leisti juos keisti. Paliekama vis daugiau laisvės pačiam interneto vartotojui, keičiant jau sukurtas internetines sistemas, adaptuojant jas savo poreikiams. Pavyzdžiui asmeninės svetainės nebėra kuriamos kaip seniau (bent jau nebe visos). Yra specialios svetainės, kurios greitai leidžia susikurti kiekvienam vartotojui savo internetinį dienoraštį (Blog) ir pan. Vartotojui leidžiama net keisti ne savo sukurtų sistemų turinį (Wikipedia). Natūralu, kad vartotojas tikisi iš internetinės sistemos greito, dinamiško veikimo. Duomenų kiekiai dideli, reikalavimai taip pat. Vartotojas nustos naudotis sistema, jei jam reikės ilgai laukti, kol sistema atliks paprastą veiksmą. Laimei technologijos sparčiai ištobulėjo ir atsirado poreikis taip vadinamoms dinamiškoms sistemoms.

Pirmiausiai ir apsibrėšime, ką mes laikysime dinamiška sistema arba dinamišku internetiniu puslapiu (toliau puslapis). Galima sakyti tai yra tas pats. Sistema gali būti dinamiška, jei bent keli jos puslapiai yra dinamiški. **Dinamiška sistema (puslapiu)** šiame darbe laikysime *sistemą (puslapį), kurios (-io) daugumos funkcijų veikimas didžiąja dalimi paremtas asinchroniniu bendravimu su serveriu arba, kitaip tariant, naudoja Ajax technologijas*. Sistemos kiekvienos funkcijos paprastai nereikia realizuoti, pasinaudojus Ajax technologija, nes tai apsunkina patį kūrimo procesą ir sistemos dizaino tvarkymą. Sistemą galima laikyti dinamiška, jei pagrindinės, dažniausiai naudojamos funkcijos yra sukurtos, naudojant Ajax galimybes. Geriausia nustatyti sistemos funkcijas, kurios realizuotas ne dinamiškai, veiktų per daug lėtai. Šias funkcijas tikslinga kiek įmanoma padaryti dinamiškesnėmis. Negalima imti tiesiog ir naudoti Ajax technologijų, bet kaip. Reikia sukurti dinamiškos sistemos (puslapio) dinamišką modelį. **Dinamiškos sistemos (puslapio) modeliu šiame darbe laikysime** *architektūrą, apibrėžiančią dinamišką sistemą ar jos puslapį.*. Sukurtas modelis padės įgyvendinti ne vieną funkciją, o iš karto kelias ar net visas sistemos funkcijas, kurios bus numatytos realizuoti, kad sistema taptų dinamiška.

Šio **magistro darbo tikslas** pagerinti dinamiškų interneto sistemų kūrimo metodus, sukuriant bazinį dinamiškos sistemos Ajax architektūros modelį; jį realizuoti, ištirti ir duoti

rekomendacijas praktiniam kūrimui bei tolesniam tobulinimui. Žinoma neįmanoma sukurti visiškai universalaus modelio. Mes orientuosimės tik į Java platformos interneto sistemas, kuriose viena iš funkcijų bus pokalbiai, vykstantys realiame laike. Tai bus mūsų **tyrimo sritis**. Mus domins tik tokios sistemos, kurias kuriant naudojamas java karkasas, bet pats Ajax nėra apvilktas jokiais pagalbiniais karkasais, kurie supaprastina jo naudojimą. Karkasas reikalingas, kad kuriama sistema būtų išbaigta ir patobulinta galėtų būti praktiškai panaudota. Mūsų **tyrimo objektas** bus sukurtas bazinis dinamiškos sistemos (puslapio) modelis, kuris bus projektuojamas, pasinaudojant Spiar architektūros modeliu. Sukurtas modelis ir jo realizacija padės išspręsti tokias **problemas ir klausimus**:

- Naujų interneto tendencijų ir ateities perspektyvų klausimas.
- Poreikio dinamiškumui klausimas.
- Ajax poveikio dizaino kūrimui klausimas.
- Realizuotos sistemos savybių tyrimo ir įvertinimo klausimai.
- Modelio tobulinimo galimybių klausimas.
- Modelio veikimo spartos problemos.
- Dinamiškos sistemos realizacijos problemos.
- Dinamiškos sistemos (puslapio) suvokimo problemos.
- Bazinio Ajax veikimo principų supratimo problemos.

Pabraukta problema yra pagrindinė. Paminėtos problemos ir klausimai suformuoja darbo uždavinius, kurie bus sprendžiami projektavimo ir kūrimo procese :

- Apibūdinti antros kartos internetą.
- Ištirti dinamiškų funkcijų realizacijos poreikį.
- Pasirinkti vieną iš dviejų kuriamo modelio serverio apklausinėjimo strategijų.
- Sukurti bazinį dinamiškos sistemos (puslapio) Ajax architektūros modelį.
- Kuo aiškiau pademonstruoti Ajax veikimo principus.
- Realizuoti sukurtą modelį pažinčių portalo sistemoje.
- Realizuoti sistemoje pokalbius realiu laiku, pasitelkiant sukurtą modelį.
- Įvertinti sistemos internetinius puslapius, sukurto modelio pagrindu bent 4 aspektais.
- Parodyti dinamiškos sistemos privalumus.
- Nustatyti modelio trūkumus ir tobulinimo galimybes.

Kadangi pasaulyje paminėti uždaviniai ir problemos yra aktualūs. Tai šių dienų problemos, kurios intensyviai yra sprendžiamos. Ajax vis labiau yra integruojamas į kitus karkasus. Kuriami ir Ajax karkasai, kurie palengvina Ajax naudojimą. Mūsų tikslas yra nesukurti naują Ajax karkasą, bet **pademonstruoti** vieną iš galimų Ajax panaudojimo galimybių, per sukurtą bazinį dinamiškos sistemos modelį.

Pagalbai kuriant modeli buvo pasitelkta jau kitų žmonių sukurta patirtis: IEEE, Wikipedia O`Reilly knygos, Ali Mesbah ir Arie van Deursen aprašytas Spiar Ajax modelis, kurio pagrindu buvo kuriamas ir mūsų dinamiškos sistemos modelis. Pagrindinę literatūrą sudaro žinomų autorių straipsnių medžiaga. Kadangi dinamiškumo problema aiški – informacijos yra labai daug. Bet informacija buvo atrinkta, atsižvelgiant į tyrimo sritį. Pasinaudojus visais paminėtais šaltiniais ir asmenine patirtimi buvo sukurtas ir realizuotas dinamiškos sistemos (puslapio) modelis. Spiar modelis buvo modifikuotas ir pakeistas taip, kad būtų priderintas prie planuojamos kurti sistemos. Pritaikymas vyko teorinių eksperimentų būdu, tikintis, kad realizacija parodys tinkamus rezultatus. Modelis buvo labai detaliai dokumentuotas, nuo architektūros iki sekų diagramų, todėl lengvai galės būti panaudotas ir suprastas kitų žmonių. Realiai buvo sukurta veikianti pažinčių portalo tipo sistema, kurioje realizuotas paminėtasis modelis.

Sistema sukurta naudojami Tapestry ir Hibernate karkasus. Sistema buvo ištirta ir nustatyti bei įvertinti jos puslapių dinamiškumo rodikliai, kurie parodė, kad sukurta sistema yra **dinamiška**. Taip pat buvo nustatyta, kad funkcijos, realizuotos pasitelkus Ajax veikia daug greičiau nei funkcijos, kurioms nebuvo naudojamos asinchroninės technologijos. Kad visa tai nustatyti buvo atliekami realūs eksperimentai, kurių metu sistemos suskaičiuoti tiriamo modelio veikimo laikai buvo nuskaityti ir įvertinti. Viskas vyko realiomis sąlygomis. Taip pat buvo vertinamas sistemos puslapių programavimo sudėtingumas, kurį įnešė Ajax naudojimas. papildomai buvo vykdomi realūs pokalbiai realiu laiku, kad pilnai ištirti modelį, kuris kaip tik ir buvo orientuotas į paminėtąją sistemos galimybę.

Gauti rezultatai yra gan kokybiški, nes laikas buvo skaičiuojamas milisekundžių tikslumu. Laikus pateikė pati sistema, nes ji buvo kuriama taip, kad galėtų pateikti visus skaičiavimus eksperimento dalyviams (šiuo atvejų – magistro darbo kūrėjui). Buvo atliekama daug eksperimentų ir skaičiuojami laiko vidurkiai. Iki dar didesnės kokybės trūko didesnio vartotojų skaičiaus. Deja techninės kliūtys ir personalo trūkumas to neleido padaryti. Tiriant realius pokalbius buvo rasti nedideli modelio trūkumai, kurie pasimatė ir eksperimentuojant su sukurta sistema. Realybė parodė, kad sukurtą modelį reikia tobulinti. Nors ir esant poreikiui modelį patobulinti – jis laisvai gali būti pritaikomas kurti realaus laiko pokalbių funkcionalumui realizuoti. Išplėtus serverio ir kliento naršyklės priimamų bei formuojamų

komandų sąrašą, galima sistemos funkcijas smarkiai išplėsti, nekeičiant visai pačio modelio, o tik papildant pagrindines jo dalis. Taip pat šį modelį galima pritaikyti nebūtinai realiems pokalbiams realizuoti.

Modelis tinka **įvairios** paskirties sistemoms kurti. Kaip pavyzdys galėtų būti namų apsaugos sistema. Prisijungęs namo savininkas galėtų realiu laiku patikrinti visus daviklius ir panašiai. Žinoma reikėtų atlikti kitus darbus, kurie nesusiję su modelio keitimu. Pati sukurta sistema taip pat yra naudinga ne vien eksperimentams ar mokymosi tikslams. Joje realizuotos visos pagrindinės pažinčių svetainės funkcijos. Reikalingas nedidelis papildomas darbas, kad šią sistemą būtų galima paleisti masiniam naudojimui. Tam reikalinga patobulinti sukurtą modelį, arba numatyti tikslų galimų vartotojų skaičių, nes modelio problema, kaip bus matyti – nedideli veikimo greičio problemos. Ajax veikimo greičio problemos yra didelė problema, kuri gali būti sprendžiama įvairiai, bet mes to nenagrinėsime.

Šis magistro darbas atitinka šiandienines tendencijas ir gali būti pagrindu programuotojui, kuris nori pradėti kurti dinamiškas sistemas. Taip pat kai kuriems šis darbas bus paskatinimas naudoti Ajax. Be to ir pati sukurta sistema yra šiuolaikiška, nes dabartiniu metu pažinčių svetainės yra vienos iš populiariausių tarp interntо vartotojų. Papildomai šis darbas naudingas ir tiems, kurie domisi Tapestry karkasu. Sistema realizuota jo pagalba. Kadangi viskas yra dokumentuota – labai nesunku perprasti jos veikimą. Detali dokumentacija padės ir sistemos (modelio) ateities tobulinimui. Atsiras naujos technologijos, tobulės esamos. Ateityje dinamiškų sistemų poreikis vis daugiau **augs**, todėl sukurto modelio negalima padėti į „lentyną “ ir užmiršti. Jį galima išplėsti ir apvilkti savu karkasu, bet šis darbas ne apie tai.

Sukurtą magistro darbą sudaro **penkios pagrindinės dalys**:

1) Susipažinimas su magistro darbu, tikslais, uždaviniais ir t.t.
2) Susipažinimas su esama situacija, technologijomis ir ateities perspektyvos.
3) Ajax dinamiškos sistemos bazinio modelio detalus aprašymas.
4) Realizuotos sistemos pristatymas ir modelio, bei sistemos puslapių tyrimų rezultatai ir įvertinimai.
5) Bendros darbo išvados.

Šis darbas, straipsnio pavidalu, buvo pristatytas „12-oji tarpuniversitetinė magistrantų ir doktorantų konferencija. INFORMACINĖ VISUOMENĖ IR UNIVERSITETINĖS STUDIJOS (IVUS‘07)“. Straipsnis pridėtas trečiame darbo priede. Darbo metu atliktos analizės rezultatai buvo panaudoti Eureka projekte: Internetinio informacinių technologijų (IT) žinių portalo sukūrimas, panaudojant vartotojų piramidinę informacijos filtracijos ir turinio formavimo technologiją (Santrumpa „IT-Europe”). Lietuvos Mokslo ir studijų fondo remiamas Eureka programos projektas „IT-Europe” (Reg. No 3473), 2005.

2. DINAMIŠKŲ SISTEMŲ (PUSLAPIŲ) POREIKIS IR KŪRIMO METODAI

2.1. Antros kartos Internetas (Web 2.0)

Kaip teigia Tim O`Reilly [1], interneto „burbulas “ sprogo 2001 metais. Šis sprogimas pažymėjo interneto pasikeitimo tašką. Web 2.0 konceptas buvo pradėtas konferencijoje kolektyviniame naujų idėjų svarstyme tarp O`Reilly ir MediaLive International [1]. Dale Dougherty, kuris yra interneto pionierius ir O`Reilly VP, pažymėjo, kad internetas tapo dar svarbesniu, nei bet kada anksčiau, reguliariai atnešdamas vis naujas svetaines ir stulbinančias aplikacijas [1]. Toliau panagrinėsime konkretesnius Web 2.0 apibrėžimus. Lietuva irgi neatsilieka nuo pasaulio ir nagrinėja, bei pastebi Web 2.0 privalumus ir naujoves. [4] šaltinis patvirtina, kad Web 2.0 termino pradininkai yra paminėtieji O`Reilly Media group. Kalbant apie pačio termino apibrėžimą susiduriame su problema. Kaip teigia [4], terminas yra įvairiai interpretuojamas ir pati jo reikšmė nėra tiksliai apibrėžta. Vieną iš apibrėžimų pateikia Wikipedia [2]. Web 2.0 – tai antros kartos internetu paremtos paslaugos: socialinės-tinklinės svetainės, wiki, komunikavimo įrankiai bei vartotojų sukurtos sistemos, kurios skirsto į kategorijas interneto puslapius (folksonomies). Folksonomy Wikipedia apibrėžia taip [2]: tai vartotojo sukurta sistema, naudojama internetinių puslapių, nuotraukų, nuorodų ir kito žiniatinklio turinio gavimui ir skirstymui į kategorijas. Sukurtos sistemos pabrėžia internetinį bendradarbiavimą ir informacijos dalinimąsi tarp vartotojų [2]. Web 2.0 stiprybė yra duomenys. Žiniatinklis tampa lyg platforma, kurios pagalba gali būti kuriamos naujos paslaugos. Kuo daugiau žmonių naudojasi paslauga – tuo didesnę naudą jie gauna. [4] šaltinio teigimu, tai atsispinti pažinčių svetainėse, kurios jau senokai neatitinka tokio primityvaus apibrėžimo. Šios svetainės dabar pirmiausiai suteikia galimybę išreikšti save. Kaip pavyzdys, pateikiama MySpace. (My Space karta). Kaip jau minėta Web 2.0 stiprybė yra duomenys. Svarbu ne tik patys duomenys, bet ir tai, kad Web 2.0 suteikia galimybę dirbti su dideliais duomenų kiekiais. Kaip pavyzdį [4] šaltinis pateikia Wikipedia, kuri pagal duomenų kiekį yra spausdintos Britannica enciklopedijos versijos varžovė. Asmeninius puslapius keičia asmeniniai dienoraščiai (blog). Tokius dienoraščius yra patogu valdyti, keisti, atnaujinti ir t.t. Taip pat [4] šaltinis teigia, kad dar viena labai svarbi Web 2.0 savybė yra nuolatinis tobulinimas (the perpetual beta). Vartotojai turi nuolat ieškoti naujų paslaugų pritaikymo būdų. Tos paslaugos gali būti net nenumatytos kūrėjų, bet jie nuolat tobulina kodą ir siūlo naujas funkcijas.

O`Reilly [1] pateikia savo Web 2.0 suvokimą palyginimą su Web 1.0 . Lyginama pasinaudojus pavyzdžiais.

Web 1.0		**Web 2.0**
Dvigubas spragtelėjimas	→	Google AdSense
Nuotraukos internete	→	Flickr – nuotraukų dalinimasis
Akamai	→	BitTorrent
Mp3.com	→	Napster
Britannica Online	→	Wikipedia
Asmeninės svetainės	→	Blogging
Evite	→	Upcoming.org ir EVDB (įvykių duomenų bazė)
Domenų vardų spėlionės	→	Paieškos variklių optimizavimas
Puslapių peržiūros	→	Kaina už paspaudimą
Ekrano valymas (scraping)	→	Žiniatinklio paslaugos (web services)
Publikavimas	→	Bendradarbiavimas
Turinio valdymo sistemos	→	Wikis
Katalogai (sistema)	→	Žymėjimai („ folksonomy“)
Keblumas	→	Susijungimai (syndication)

Kaip jau buvo minėta Web 2.0 nėra apribotas griežtomis konstantomis, reikalavimais ir taisyklėmis, bet galime apibrėžti kelias pagrindines charakteristikas, kurias pateikia [2] (Wikipedia). Taigi, kol yra diskutuojama dėl Web 2.0 aplikacijos apibrėžimo, galime apibrėžti kelias bazines Web 2.0 charakteristikas, kurios galėtų būti [2] :

- „Tinklas kaip platforma “ – aplikacijos pateikiamos pilnai per naršyklę ir taip leidžiant vartotojams jas naudoti.
- Vartotojai valdo savo duomenis, kurie yra svetainėje ir vykdo tų duomenų kontrolę..
- Bendradarbiavimas ir demokratija skatina vartotojus pridėti vertę jų naudojamoms aplikacijoms. Tai yra didelė priešingybė hierarchiniam valdymui, kuriame sistemos priskiria vartotojams roles su kintančiais funkcionalumo lygiais.
- Puikus, interaktyvus, „vartotojams-draugiškas „ vartotojo interfeisas, paremtas Ajax ar panašiais karkasais.

- Kai kurie socialinio-tinklo aspektai. Wikipedia pateikia tokį socialinio-tinklo apibrėžimą. Tai socialinė struktūra, sukurta iš mazgų (dažniausiai individai arba organizacijos), kuriuos jungia vienas ar daugiau specifinių ryšių, tokių kaip finansiniai apsikeitimai, draugystė, prekyba, žiniatinklio nuorodos ir panašiai.
- Vieša nauda, kuri paprastai turi du parametrus- sutelktinis tiekimas ir neišsiskiriamumas.

Prie Web 2.0 charakteristikų galima būtų priskirti ir technologijas, kurias Web 2.0 svetainė paprastai tūri naudoti. [2] šaltinis pateikia tokį sąrašą metodikų-technologijų:

- Turtingos (rich) internetinės taikomosios programos, neprivalomai paremtos Ajax.
- CSS (Cascading Style Sheets).
- Semantiškai teisingos XHTML žymės ir mikro-formatų naudojimas. (mikro-formatų žymės tai specifiniai HTML atributai: class, rel, rev).
- RSS/Atom viduje esančių duomenų surinkimas ir įgalinimas dalinti (-is) ją.
- Aiškūs ir suprantami URL.
- Didelis folksonomies naudojimas (pvz. žymių forma).
- Dalinis arba pilnas wiki programinės įrangos naudojimas (dalinis naudojimas gali išaugti iki pilnos platformos, skirtos svetainei).
- Internete sugeneruotų (webblog) svetainių publikavimas.
- REST arba XML interneto paslaugų APIs.

Apžvelgę charakteristikas ir technologijas, detaliau aptarkime kelias naujoves, kurios yra susijusios su nagrinėjamu Web 2.0. [2] šaltinis pateikia tokias naujoves:

- **Internetu-paremtos aplikacijos ir darbastaliai**. Ajax suteikta turtingesnė vartotojo-patirtis paskatino svetainių kūrimą, kurios imituoja personalinio kompiuterio aplikacijas. Tai žodžių apdorojimo, prezentacijų ir kitos. WYSIWYG wiki svetainės replikuoja daugybę, personaliniams kompiuteriams sukurtų programų, savybių. Vis dėlto, kitos svetaitinės atlieka bendradarbiavimo ir projektų valdymo funkcijas. Pvz. Google Inc. Taip pat atsirado ir naršykle paremtų operacinių sistemų ir internetinių darbastalių, tik jos daugiau funkcionuoja kaip platformos.
- **Turtingesnės interneto aplikacijos.** Ajax, Adobe Flash, Flex ir OpenLazlo technologijos ištobulina vartotojų patirtį naudojantis naršyklėmis paremtomis aplikacijomis. Paminėtos technologijos leidžia internetiniui puslapiui prašyti

atnaujinti kai kurias jo turinio dalis, atnaujinti tas dalis naršyklėje be pačio puslapio perkrovimo.

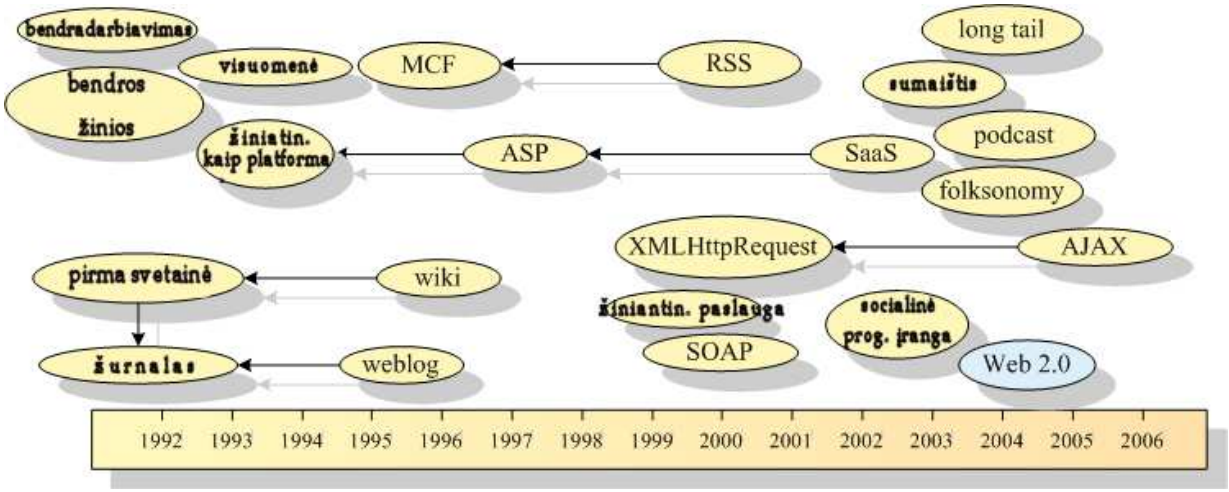
- **Serverio-pusės programinė įranga.** Papildomas Web 2.0 suteiktas funkcionalumas priklauso nuo vartotojų galimybių dirbti su duomenimis, kurie yra išsaugoti serveriuose. Tai jie gali padaryti: HTML formų pagalba, raštų (script) kalbos pagalba (pvz. JavaScript) arba pasinaudojus Flash ar Java technologijomis. Visos paminėtos technologijos naudoja kliento kompiuterio resursus. To pasekoje sumažėja serverių darbo krūvis.
- **RSS.** Kai kurie standartizuoti protokolai leidžia galiniam vartotojui naudotis svetainės duomenimis visai kitame kontekste. Kontekstas gali būti kita svetainė, naršyklės įskiepis ar net atskira darbastalio aplikacija. Vienas iš protokolų, kuris leidžia tokius dalykus yra RSS (Really Simple Syndication – taip pat žinomas kaip „web syndication “).
- **Internetinia protokolai.** Žiniatinklio komunikavimo protokolai suteikia svarbiausius Web 2.0 infrastruktūros elementus. Svarbesni protokolai naudoja REST ir SOAP. **REST** (Atstovaujamasis būsenos perdavimas) – nurodo duomenų prieigos ir manipulacijos serveryje būdą, naudojant HTTP GET, POST, PUT ir DELETE komandas. **SOAP** – XML žinučių ir užklausų siuntimą serveriui.

Galiausiai lieka aptarti Web 2.0 trūkumus (kritiką) ir kas laukia mūsų toliau. Kaip tobulėja ir tobulės Web 2.0. [2] šaltinis pateikia tokias pastabas dėl Web 2.0 :

- Kadangi Web 2.0 nėra aiškiai apibrėžtas standartų aibe ir pan. – skirtingi žmonės Web 2.0 gali iš esmės suprasti skirtingai.
- Dauguma Web 2.0 idėjų jau buvo įgyvendintos tinklų sistemose dar prieš pačio „Web 2.0 „ termino pasirodymą. Pvz. Amazon.com jau leido vartotojams rašyti apžvalgas ir vartotojo gidus nuo pat Amazon paleidimo 2005 metais.
- Kai kurios svetainės tik paskelbia save esančias „Web 2.0 “, o iš tiesų tai tebūna rinkodaros madingas-gražus posakis.
- Taip pat pateikiamas argumentas, kad pats „Web 2.0 „ nepateikia naujos žiniatinklio versijos, o tiktai toliau naudojas „Web 1.0 “ konceptus ir technologijas. Neabejotinai tokios technologijos kaip Ajax nėra esminio HTTP protokolo pakaitalas, o tiktai papildomas sluoksnis ant jo viršaus.
- Sakomas, kad per daug Web 2.0 kompanijų mėgina sukurti tokį patį produktą, kuriam trūksta biznio modelių.

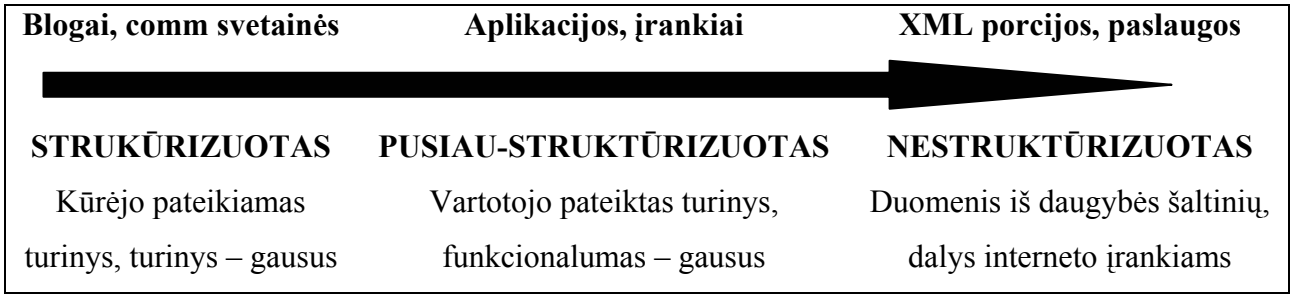
- Dauguma vartotojų naudoja Web 2.0 aplikacijas (Wikipedia ar MySpace) be jokio poreikio aiškiai suprasti „Web 2.0 „ terminą.

Lieka aptarti Web 2.0 tobulėjimą ir plėtimąsi.



1 pav. Web 2.0 atsiradimas bendroje laiko skalėje [2].

Kaip teigia Dan Saffer [3], visi daugiau kalba apie tai, ką reiškia „Web 2.0 “, o ne kokią įtaką tai duoda vartotojo patirčiai. Dan Saffer pateikia tokius pasikeitimus.



2 pav. Web 2.0 plėtojimasis

Trumpai pirmoje dalyje turime gerai žinomas svetaines, kaip blogai, namų puslapiai, prekybos ir bendravimo svetainės, paieškos varikliai ir t.t. Tai strukūrizuotas sistemos plėtimasis, kurios forma ir turinys yra daugiausiais nustatomas jų projektuotojų ir kūrėjų [3].

Per vidurį turime turtingas, darbastalio-tipo taikomąsias programas, kurios dėka Ajax, Flex, Flash, Laszlo ir t.t. yra migravusios į žiniatinklį. Tai gali būti standartinės žodžių apdorojimo, skaičiavimų, elektroninio pašto ir kitos. Bet labiausiai įdomios yra internetu-paremtos aplikacijos, kurios išnaudoja interneto privalumus. Tai Yahoo Groops, blogs, Flickr ir kitos. Plėtimasis nustato sistemų tipus, kuriuos vartotojai galės turėti, bet vartotojai pateikia savo turinį [2].

Galiausiai paskutinėje dalyje turime nestruktūrizuotas sistemas – naujų paslaugų perpildymą. Dauguma paslaugų iš vis nereikalauja lankyti svetainės. Laisvi aplikacijų dalių,

turinio ir duomenų rinkiniai, kurie realiai niekur neegzistuoja, tačiau gali būti surasti, panaudoti, panaudoti iš naujo, sutvarkyti ir sumaišyti. Bet tam reikalingi įrankiai, nes paprastai HTML be naršyklės neturi jokios prasmės. Web 2.0 reikalauja naujų žiniatinklio naudojimo būdų. Naudojimas neturi būti tik paprastas naršymas (navigavimas), kad panaudoti visų įgūdžių privalumus [3]. Kadangi viskas labai greitai tobulėja – sunku spėti pasivyti, bet pagal Dan Saffer [3] niekas neatsiranda per naktį, kai kurie nauji dalykai tampa jau praeitimi. Taigi pagal [3] šaltinį po dešimties metų (simboliškai, tiesiog ateityje) bus atsisakyta keltas sąvokų, kurių jau nebebus naudojamos. Tai galėtų būti svetainės, žymekliai (bookmarks) ir t.t. Taip pat teigiama, kad jau netolimoje ateityje mums nebereikės adresų, kad galėtume užsisakyti paslaugas. Tai bus padaroma taip vadinamųjų widgets (nedideli paruošti komponentai) pagalba.

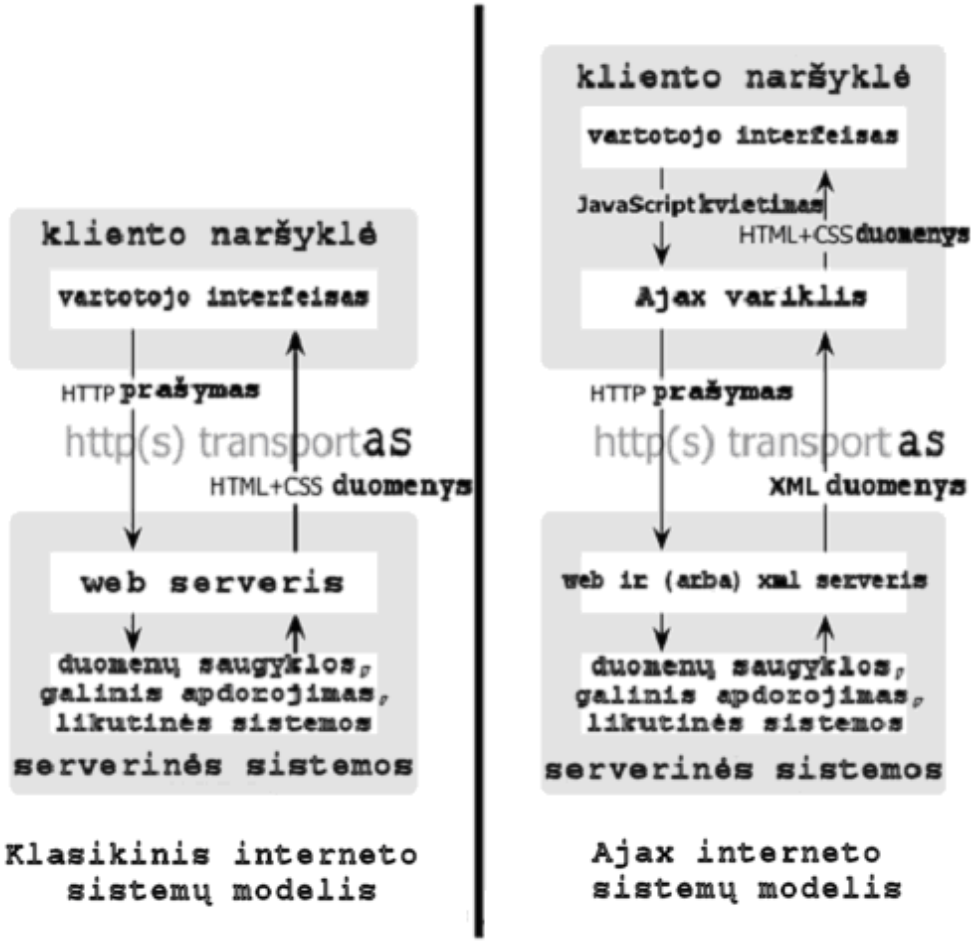
Taigai aptarėme žiniatinklio perspektyvas ir naujoves. Pamatėme, kad viskas kita greitai, dinamiškai. Žmonės dažnai nespėja su technologijomis. Dažnai niekas nesusimąsto kaip kas veikia, bet žino, kad nori greitai veikiančių, dinamiškų ir gražių internetinių puslapių. Web 2.0 pateikė aibę technologijų, leidžiančių tai padaryti. Daugumoje šaltinių neapsieinama nepaminėjus Ajax technologijų rinkinio, karkaso ir apskirtai Ajax termino. Sekančiame skyriuje aptarsime šį terminą ir bandysime išsiaiškinti kodėl jis yra toks svarbus dinamiškų puslapių ir apskirtai jų kūrime.

2.2. AJAX (Asinchroninis JavaScript ir XML)

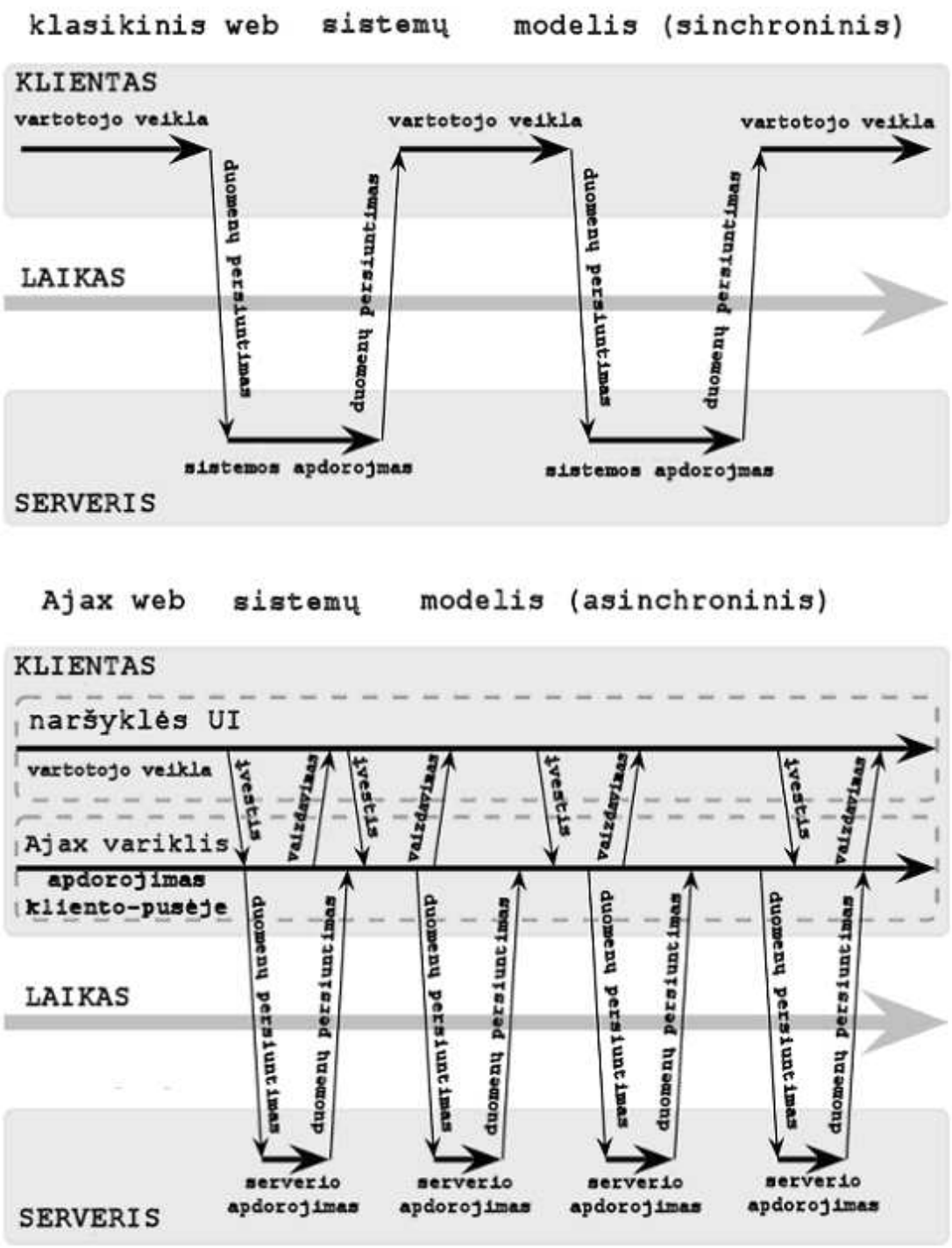
2.2.1 Architektūra, veikimo principai

Asynchronous JavaScript and XML, arba **Ajax**, yra žiniatinklio plėtojimo technika, skirta kurti interaktyvioms interneto aplikacijoms [10]. Realiai Jesse James Garrett [11] teigia, kad Ajax nėra tik technologija. Realiai tai yra keletas technologijų, kur kiekviena iš jų veikia savaip, bet kartu sudėjus gali būti panaudota labiau efektyvesniems tikslams.
Ajax apjungia tokias technologijas:

- XHTML (or HTML) ir CSS informacijos žymėjimui ir stilizavimui.[10]
- Dokumentų Objektų Modelis (DOM) manipuliuojamas panaudojant JavaScript, kad dinamiškai atvaizduoti pristatomą informaciją ir sąveikauti su ja.[10]
- XMLHttpRequest objektas, naudojamas asinchronimiam duomenų pasikeitimui su serveriu.[10]
- Duomenų mainams ir manipuliacijoms naudojama dar XML ir XSLT. [11]
- JavaScript, kuris jungia viską į vieną krūvą. [11]



3 pav. Klasikinės ir Ajax internetinių sistemų modeliai [11]



4 pav. Sinchroninės ir asinchroninės (Ajax) sąveikų palyginimas [11]

Klasikinis Web aplikacijos modelis dirba taip. Dauguma vartotojų veiksmų interfeise sužadina HTTP užklausą atgal į web serverį. Tada serveris atlieka kažkokius veiksmus ir grąžina rezultatus į į HTML puslapį pas klientą. Šis technologinis sprendimas gal ir turi techninę prasmę, bet vartotojas turi laukti, kol serveris apdoros užklausimą [11]. Pavyzdžiui, užpildžius anketą (ar formą), kreipiamasi į serverį ir užkraunamas naujas puslapis. Tokiu būdu bereikalingai iššvaistomi resursai, nes didelė dalis informacijos nepakinta ir siunčiama kaskart

iš naujo. Taip pat tokiu būdu neįmanoma pasiekti tokio interaktyvumo, kokį gali suteikti ne internetinės aplikacijos [10].

Kyla klausimas, kam vartotojui iš viso matyti, kaip taikomoji programa kreipiasi į serverį iš vis. Štai šioje vietoje ir turime Ajax, kuris eliminuoja pradėti-sustoti-pradėti-sustoti sąveikos prigimtį internete bei pristato savo Ajax variklį tarp vartotojo ir serverio. Vietoj to, kad užkrauti internetinį puslapį sesijos pradžioje, naršyklė užkrauna Ajax variklį, kuris parašytas JavaScript kalba ir paprastai paslepia jį slaptame rėmelyje (frame). Variklis yra atsakingas už interfesio atvaizdavimą, kurį mato vartotojas ir komunikavimą su serveriu vartotojo naudai. Ajax variklis vartotojui leidžia sąveikauti su aplikacija asinchroniškai ir nepriklausomai nuo komunikacijos su serveriu. Taigi vartotojas niekada nelaukia, žiūrėdamas į tuščią ekraną, kol serveris kažką daro. Kiekvienas vartotojo veiksmas vietoj to, kad paprastai sugeneruos HTTP užklausą – perduos JavaScript formos šaukinį (call) į Ajax variklį. Bet koks atsakas į vartotojo veiksmą nereikalauja sugrįžimo į serverį (pvz., paprasta laukų reikšmių patikra). Ajax variklis pats viską sutvarko. Jei varikliui reikia ko nors iš serverio, kad atsakyti į vartotojo veiksmą, jei tai yra duomenis, siunčiami apdoroti, reikalingas užkrauti papildomas interfeiso kodas, ar naujų duomenų išgavimas – variklis šias užklausas vykdo asinchroniškai, naudodamas XML, neuždelsiant vartotojo sąveikos su aplikacija [11]. Ajax svetainės gali siųsti užklausas serveriui, gauti atsakymą tam tikra apsibrėžta forma (SOAP ar kita XML paremta), bei naudojant JavaScript programavimą atnaujinti tik reikiamą puslapio dalį. Tokiu būdu sutaupomi tinklo resursai, nes žymiai sumažinami duomenų srautai. Taip pat taupomas ir serverio procesoriaus resursai, nes dalis logikos perkeliama klientui.

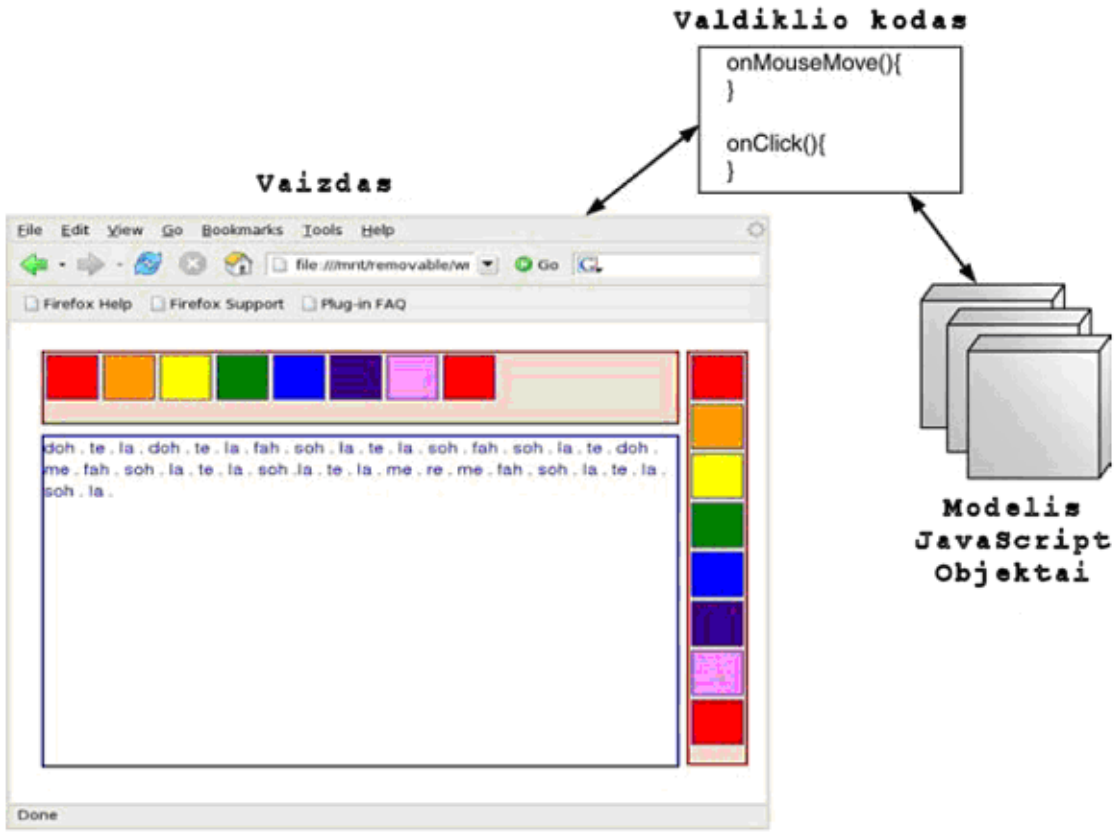
Puslapiai, sukurti naudojant AJAX technologija, reikalauja naršyklių, palaikančių šias technologijas. Tokios naršyklės yra Mozilla Firefox, Internet Explorer, Opera, Konqueror ir Safari [10].

Ajax jau yra gan plačiai pradedamas naudoti. Google deda dideles investicijas į Ajax vystymą. Orkut,Gmail, naujausia versija google Groups, Google Suggest ir Google maps yra Ajax taikomosios programos. Flickr priklauso nuo Ajax, Amazon A9.com paieškos variklis naudoja panašias technologijas [11].

Ajax nesugadina tradicinio modelis-vaizdas-valdiklis kūrimo principo, kuris yra dažnai taikomas internetinių puslapių kūrime. Tai toks principas kai duomenys yra atskiriami nuo jų vaizdavimo logikos ir juos pasiekti galima tiki per valdiklį. Todėl toliau trumpai panagrinėsime Ajax iš Mvc perspektyvos.

1 lentelė. Ajax ir MVC

Komponentas	Aprašymas
Modelis	Vidinė būsenų reprezentacija – paspaustas, nepaspaustas, aktyvus, neaktyvus kaip pavyzdys yra modelis. Ajax tipiškai įgyvendins modelį kaip JavaScrip objektą [12].
Vaizdas	Dokumento Objekto Modelio (DOM) mazgai, Ajax vartotojo interfeiso atveju kartu su modifikacijomis sudarys vaizdą [12].
Valdiklis	Vidinis kodas, siejantis modelį su vaizdu ir bus valdiklis. Įvykių-prižiūrėtojo kodas (kas nutinka, kai didelėje aplikacijoje vartotojas paspaudžia mygtuką) taip pat yra valdiklis, bet ne valdiklis modeliui ir vaizdui [12].



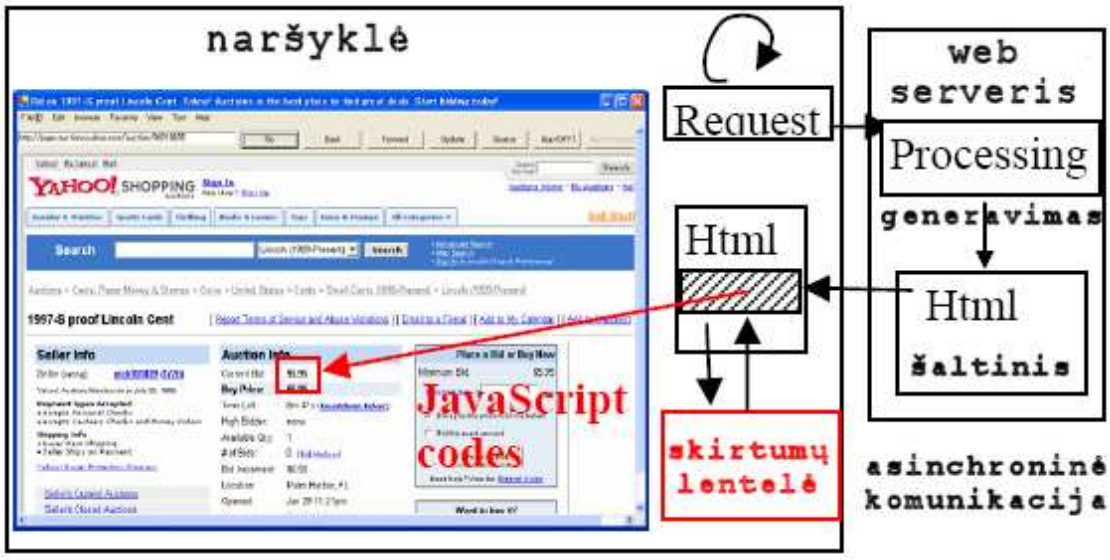
5 pav. MVC Ajax [12]

Kol kas buvo kalbama apie Ajax iš internetinių puslapių perspektyvos. Tai natūralu, nes norime kurti dinamiškus puslapius, kurie padėtų vartotojui patogiau, greičiau ir gražiau pateikti informaciją ir suteikti galimybę manipuliuoti ja (Web 2.0). [7] straipsnio autoriai pažvelgia į Ajax visai iš priešingos pusės. Ajax technologijas turi palaikyti kliento naršyklė, todėl natūralu

sukurti naršyklę, kuri jau turėtų Ajax savyje arba dirbtų tokiais pačiais principais. Apie tai ir bus kalbama sekančiame skyrelyje.

2.2.2 Naršyklės modelis Ajax pagrindu

Kaip jau žinoma, žiniatinklio aplikacijas pagrindinai yra pasiekiamos naršyklės pagalba. [7] straipsnio autoriai teigia, kad dėl sinchroninio naršyklės komunikavimo su serveriu, internetinių puslapių veikimas naršyklėje nėra geresnis už darbastalio aplikacijų veikimą. Todėl [7] straipsnio autoriai siūlo Ajax žiniatinklio naršyklę su asinchroniniu komunikavimo modeliu. Kaip teigia šios naršyklės kūrėjai – jų naršyklė gali padidinti naršyklės veikimą iki darbastalio aplikacijų lygio, be jokių žiniatinklio aplikacijų kodo pataisymų. Tai padaroma su naršyklės nauja savybe atnaujinti dalį internetinio puslapio net ir jei internetinis puslapis nenaudoja Ajax. Kaip teigia autoriai jie jau gavo patvirtinimą, kad su jų naršykle (prie didesnių serverio apkrovų) „Yahoo auction sites „ veikė greičiau.



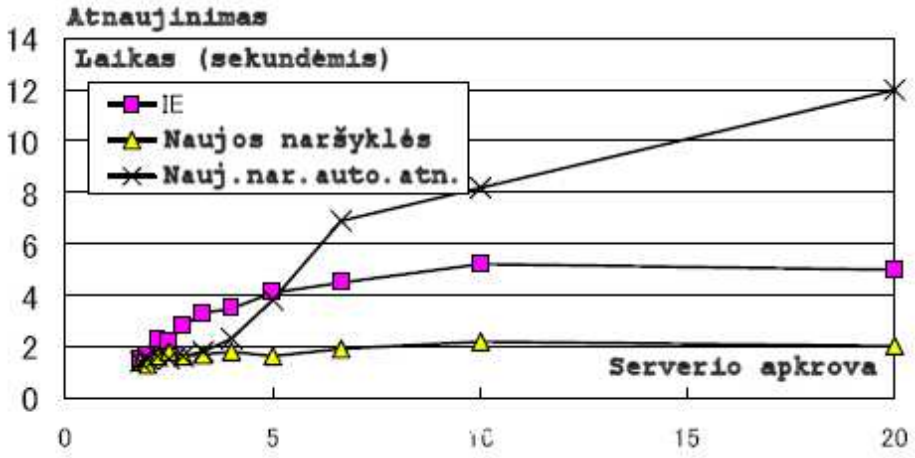
6 pav. Bazinė naršyklės koncepcija [7]

Visa asinchroninio komunikavimo modelį sudaro 4 dalys, kurios gali ir naujos naršyklės internetinių puslapiui apdorojimui žingsniai:

- **1 dalis: gramatinis nagrinėjimas (parsing).** HTML šaltinis (source) yra nagrinėjamas analizatoriaus, kurio veikimo pagrindas grįstas HTML žymėmis. Taip sukuriamas sintaksės medis, kur mazgai reiškia HTML žymes, o lapai – tekstinį turinį.
- **2 dalis: JavaScriptGeneravimas.** Orginalus HTML šaltinis neturi funkcijų daliniui atnaujinimui. JavaScript kodas turi ryšį su 3 dalyje sugeneruota skirtumų lentele.
- **3 dalis: skirtumų ieškojimas.** Lyginami du sintaksės medžiai. Tokiu būdų aptinkami internetinių puslapių skirtumai. Jei reikalingas dalinis puslapio atnaujinimas –

generuojama skirtumų lentelė, kurioje nurodomi skirtumų mazgų informacija. Tai pažymėjimo (tag) identifikatorius, tekstai, savybės.

- **4 dalis: dalinis atnaujinimas.** Remiantis skirtumų lentele, esamas puslapis prideda JavaScript kodą, kuris atliks dalinį atnaujinimą. Norint, kad puslapis atsinaujintų dalinai, jis kreipiasi į skirtumų lentele pastoviais intervalais. Pagrindinis blokas vykdomas nepriklausomai nuo vidinio bloko, kuris vykdo komunikaciją su serveriu.



7 pav. Naršyklės veikimo sparta [12]

Kūrėjų vykdomų eksperimentų eigoje buvo gauti tokie spartos rezultatai. Kaip ir teigiama [12], naujos naršyklės sparta prie didesnio serverio aprkovimo yra didesnė negu įprastinės IE naršyklės. Nepalyginami duomenys su kitomis naršyklėmis, kurių greičiai gali būti didesni nei IE.

Atsiradus naujai technologijai paprastai visi stengiasi ją įsisavinti ir paprastai tai padarę kuria patogesnes priemones tai technologijai naudoti. Ne išimtis yra ir Ajax. „Pliką " Ajax naudoti yra gan nepatogu (bet šiame magistro darbe bus ir toks panaudojimas), nes reikia rūpintis duomenų perdavimu, formatu, jų išskyrimu ir valdymu. Patogesniam naudojimui yra sukurti Ajax karkasai, kurie palengvina ir paspartina kūrėjų darbą. Šiame magistriniame darbe nebus naudojamas joks Ajax karkasas su tikslu kuo aiškiau pademonstruoti šios technologijos privalumus ir veikimo principus.

2.2.3 Ajax karkasai

2 lentelė. Ajax karkasai

Karkasas	Aprašymas
Echo2	Tai atviro kodo Ajax karkasas, kuris leidžia kūrėjui, naudojant, objektiškai orientuotas, UI komponentais paremtas ir įvykiais paremtas internetines kūrimo paradigmas, sukurti internetines aplikacijas.Tai Java aplikacijos

	karkasas, kuris suteikia API (UI komponentai, savybių objektai ir įvykių klausytojai), kurio pagalba galima pavaizduoti ir valdyti vartotojo interfeisus ir aplikacijos būsenas. Visas komponentų atvaizdavimo ir komunikacijos su kliento naršykle funkcionalumas yra patalpintas atskirame modulyje, pavadintame Web Rendering Engine. Šis modulis susideda iš Java/J2EE ir kliernto-pusės JS dalių. Echo2 turi atnaujinimų valdytoją (Update Manager), kuris yra atsakingas už vartotojo sąsajos komponento modelio atnaujinimų sekimą ir įvesties apdorojimą, gautą iš atvaizdavimo agento bei perduoti įvesti kitiems komponentams. Echo2 kliento variklis (Client Engine) veikia kliento naršyklėje ir suteikia serverio pusės aplikacijai nuotolinį vartotojo interfeisą Variklio pagrindinė veikla yra sinchronizuoti kliento/serverio būseną, kai vartotojas atlieka veiksmus interfeise. Būsenų perdavimams naudojama XML formato klientinė žinutė. Serveris atsako, suformuodamas serverio žinutę, kurioje yra daliniai atnaujinimai puslapiui (DOM) ir ją nusiųsdamas klientui [6].
GWT	Google turi orginalų požiūrį savo Ajax karkase. Google Web Framework. Kaip ir Echo2, GWT palengvina vartotojo interfeisų kūrimą, panašiu į AWT ir Swing. GWT atkeliauja su sukurtų komponentų, priemonių rinkinių bibliotke. (Widgets). GWT unikalumas išryškėja kliento pusės vartotojo interfeisų atvaizdavimo būde.Vietoj to, kad laikyti UI komponentus serveryje ir nukreipti būsenų pasikeitimus, GWT sudeda visus Java UI komponentus į JavaScript koda. Tai padaroma kompiliavimo metu. Taip pat kūrėjas turi galimybę naudoti Java 1.4 API. GWT naudoja nedidelį bendrą klientinį variklį. Naudojant kompiliatorių visas UI funkcionalumas klientinėje dalyje tampa prieinamas vartotojui. Serveris yra sutrukdomas, jei reikalinga neapdorota (raw) informacija, kad paleisti kliento komponentus. Serveris tai padaro iškviesdamas nustatytąsias paslaugas (Web Services). Tos paslaugos yra įgyvendintos Java kalba ir duomenys yra persiunčiami tinklu, pasinaudojus serializavimo (serialization) principais [6].
Backbase	Backbase yra Amsterdame įsikūrusi kompanija, kuri pateikė vieną iš pirmųjų komercinių Ajax karkasų. Šis karkasas vis dar yra tobulinamas ir yra naudojamas gausios aibės žmonių visame pasaulyje. Pagrindinis karkaso elementas yra Back-Base pristatymo kliento variklis, kuris yra parašytas JavaScript kalba ir veikia naršyklėje. Variklis yra paremtas standartais. Jis gali būti programuojamas per deklaratyvią vartotojo kalbą, kuri vadinama BXML. Serverinė Backbase karkaso puse yra formuojama Backbase Java serverio, kuris yra sukurtas ant JasaServerFases pagrindo. Backbase pateikia savo nuosavą aibę UI komponentų ir

	išplėčia JSF karkasą, kad suteikti vieno puslapio interfeiso įgyvendinimą [6].
Atlas	Atlas tai Microsoft ASP.NET nuo platformos ir našyklės nepriklausantis karkasas, skirtas kurti interaktyvioms, turtingoms, personifikuotoms interneto svetainėms. Atlas nustato pagrindinius savo tikslus : ▪ Naršykle paremtoms aplikacijoms sukurti aukškto produktyvumo platformą. ▪ Padaryti Ajax stiliaus žiniatinklio aplikacijas lengvai palaikomas, tvarkomas ir autorizuojamas. ▪ Pateikti Microsoft platformai ištobulėjusius vartotojus ir kūrėjus [6].

Net ir labai išgirta technologija turi savo blogąją pusę. Sunku sukurti visapusiškai naudingą technologiją, nes jei nori daugiau funkcijų – sudėtingėja kodo rašymas, jei norima gražesnių langų – reikia daugiau resursų ir t.t. Todėl toliau aptarsime blogąsias ir gerąsias Ajax puses.

2.2.4 Ajax privalumai ir trūkumai

AJAX privalumai:

- Ajax leidžia Web aplikacijoms atsiliepti daug greičiau į daug skirtingų vartotojo užklausų ir išvengti pakartotino nepasikeitusios informacijos siuntimo tinklu. Kadangi Ajax technologijos yra atviros, jos yra palaikomos visų naršyklių, kuriose veikia JavaScript, nepriklausomai nuo operacinės sistemos tipo [10].
- Nuolatinė komunikacija su serveriu [9].
- AJAX yra suderinamas su daugelio tarpinės programinės įrangos apibrėžimais (defininion) [9].
- Panaikinami pilni puslapių atgaliniai siuntimai (post-back), pritaikius mažesnius, didėjančius atnaujinimus [8].
- Turi įtakos kliento mašinos apdorojimo greičiui (galiai), leidžiant naršyklei būti atsakingu už daugiau taikomosios programos vykdymo aspektų [8]. Taip nuimama apkrova nuo serverio.
- Išnaudojamos šiuolaikinės grafikos galimybės: permatomumas, šešėliavimas, animacija, objektų gylio nustatymai ir t.t. Suteikiama daugiau interaktyvumo informacijos pristatymui. [8]

AJAX trūkumai:

- JavaScript kalbos apribojimai [8].
- Naršyklių nesuderinamumai [8].

- Viena didelė problema yra Ajax naudojime, kad numatyto sistemos elgesio sulaužymas dėl naršyklės „ back“ mygtuko. Vartotojo paspaudę „back“ mygtuką dažniausiai galvoja, kad internetinė aplikacija panaikins jų paskutinius pakeitimus, bet deja Ajax aplikacijose tai nebūtinai atsitiks. Žinoma egzistuoja sprendimo keliai keli šiai bėdai panaikinti [10].
- Problema, susijusi su puslapio pažymėjimu (bookmark), kadangi puslapis tampa labai dinamiškas [10].
- Tinklo gaišaties laikas arba intervalas tarp vartotojo užklausos ir serverio atsakymo turi būti atidžiai apgalvotas Ajax aplikacijose. Vartotojai gali nesuprasti , kodėl nieko neįvyko ar kodėl kažkas įvyko ne tuo laiku, kad jie to tikėjosi. Ypač jei tai susiję su vartotojo sąsajos pakeitimais [10].
- Kaip ir DHTML svetainėms, AJAX svetainėms reikia žymiai daugiau testavimo, nes kiekviena naršyklė gali elgtis šiek tiek skirtingai [10].
- Naudojant AJAX, padaugėja mikro-užklausų į serverį, tai neapgalvotai naudojant gali sulėtinti bendravimą tarp kliento ir serverio [10].
- Ajax nepritaikytas žmonėms su negalia, nes yra orientuotas tik į grafinę sąsają turinčias naršykles [10].

Kadangi nemažai diskusijų ar naudoti ar nenaudoti Ajax [13] šaltinis nurodo 10 situacijų, kada naudonga naudoti Ajax ir žinoma, kada nenaudinga.

Kada patartina naudoti Ajax:

- **Su formomis susijusios sąveikos** - formos yra lėtos, dažnai labai lėtos. [13]
- **Gili hierarchinė medžio tipo navigacija** [13]
- **Dažna vartotojas-vartotojas komunikacija** [13]
- **Balsavimas. Taip/Ne klausimai, reitingų pasikeitimai** [13]
- **Filtruojant ir susijusių duomenų manipuliacija.** [13]
- **Paprastai įvesti tekstiniai užuominų (hints) tekstai, automatinio įvedimo pagalbos tekstai**
- [13]

Taip pat pateikiama ir kada nereiktų naudoti Ajax:

- **Paprastos formos** [13]
- **Paieška** [13]
- **Standartinė pirminė navigacija** [13]

- **Vaizdavimo manipuliacijos** [13]
- **Didelio teksto kiekio pakeitimas** [13]
- **Be prasmės ir naudos papildomi pagražinimai** [13]

Kadangi Ajax apjungia aibę technologijų – reikia surasti priemonę, kuri padėtų suvaldyti tas technologijas. Be to pats vienas Ajax negalės sukurti viso puslapio, reikės papildomai naudoti ir kitas technologijas, kurios galėtų paprastai ir lengvai atvaizduoti visą reikiamą informaciją. Ajax tik priemonė padaryti puslapius labiau dinamiškus. Taigi sekančiame skyriuje bus aptartas pasirinktas karkasas, kurio pagalba buvo realizuota kuriama sistema.

2.3. Ajax ir Tapestry karkasas

2.3.1 Tapestry karkasas

Pirmiausiai trumpai apžvelgsime, kas tai yra Tapestry karkasas :

- Atviro kodo karkasas, paremtas komponentiniu kūrimu, skirtas žiniatinklio sistemoms kurti [14].
- Karkasas internetinių sistemų kūrimą padaro panašų į standartinių GUI (**g**raphical **u**ser **i**nterface) kūrimą [14].
- Suteikia galimybę kurti web aplikacijas nesirūpinant centriniu operacijų servletu [14].
- Cituojant Tapestry namų puslapį : “ Tapestry suderina internetinių aplikacijų kūrimą su objektų, metodų ir savybių terminais, vietoj URL ir užklausų parametrų „ [14].
- Tapestry yra daugiau nei tik paprasta šablonų sistema. Ji yra tam, kad kurti dinamiškas, interaktyvias svetaines [15].
- Tapestry yra realus karkasas, sudėtingų aplikacijų kūrimui, naudojant paprastus, daugkartinio panaudojimo komponentus [15].
- Tapestry yra tampriai integruota su OGNL (**O**bject **G**raph **N**avigation **L**anguage). OGNL yra Java išraiškų kalba, kuri naudojama žvilgtelėti į objektus ir nuskaityti arba atnaujinti jų savybes [15].

Pereikime prie išskiriamų Tapestry privalumų:

- **Paprastumas** – Tapestry aplikacija turi žymiai mažiau kodo nei tradicinės Servlet aplikacijos. Elimiminuojama didelė dalis „neįdomaus“ kodo, kuriuo pasirūpina pats karkasas. Nebereikia gramatiškai nagrinėti užklausų parametrų, manipuliuoti *HttpSession* objektu, kurti URL ir t.t. Leidžia kūrėjams kurti aplikacijas panaudojant būsenas

išsaugančius Java Bean, vietoj to nedarančių Servletų. Leidžia kūrėjams leisti laiką, rašant aplikacijos specifinę logiką [14]

- **Aiškumas** – suteikia aiškų požiūrį į web aplikacijos puslapių vystymą – kūrimą. Padeda pašalinti spėliones, kurios atsiranda dirbant su tradicinėmis Servlet aplikacijomis. Visi puslapiai Tapestry aplikacijoje dirba panašiai, nes yra sukurti, panaudojant tuos pačius daukartinio panaudojimo komponentus [14].
- **Efektyvumas** – Tapestry aplikacijos smarkiai keičiamo mastelio. Taip pat Tapestry savyje įgyvendina panaudojimo buferius ir objektų telkinius (pool), kadminimizuoti kiekvienos užklausos apdorojimo laiką. Tapestry aplikacijos savo sparta yra panašios į tradicines Servlet aplikacijas. Kūrėjo efektyvumas didėja, padidinus jo produktyvumą [14].
- **Grįžtamasis ryšys** – kodo eilutės tikslumo klaidų pranešimai. Nebereikia stebėti krūvas trasų, kad nustatyti nustatymų bylų klaidas. Nepagautos išlygos raportuojamos, turint visą krūvą išlygų trasų ir kiekvienos išlygų savybių. Karkasas suteikia daug grįžtamo ryšio įvykus klaidoms, net nenaudojant klaidų seklio (debugger) [14].

Susipažinus su naudojamu karkaso pagrindinėmis idėjomis bei privalumais svarbu žinoti ir kodėl jis buvo pasirinktas, o ne koks kitas atviro kodo Java karkasas.

2.3.2 Tapestry karkaso pasirinkimo pagrindimas

Tapestry 4.1 yra pilnai integruotas su Ajax technologijomis. Integruotas Dojo Javascript įrankių rinkinys pateikia jau sukurtus Ajax tipo komponentus, kuriuos galima labai lengvai panaudoti puslapiuose. Taip pat pridėta XHR – populiaraus xml komunikavimo formatas, kuris leidžia lengvai kurti bendriausias Ajax koncepcijas. Taip pat pridėtas naujas komunikavimas per JSON (JavaScript Object Notation), apie kurį bus kalbama 2.4 skyrelyje. Pridėtos naujos anotacijos Ajax funkcionalumo palaikymui bei paprastam naudojimui. Pvz. EventListener, kuris gali aptikti bet kokį DOM įvykį ir nusiųsti reikiamas užklausas į serverį. Taip pat sudaryta asinchroninė formų pateikimo (submit) galimybė. Taip pat iš senesnių versijų yra atkeliavęs XTile komponentas, skrtas Ajax asinchroniniams kreipiniams į serverį, bei atsakymų iš serverio apdorojimui [16]. Taip pat Tapestry yra pasirinktas dėl jo komponentinio puslapių kūrimo būdo. Tokiu būdu labai patogu realizuoti dinamiškus puslapius. Dar galima pridurti tai, kad Tapestry pagreitina puslapių užkrovimą, įkeldamas juos į atminti, todėl po pirmo panaudojimo internetiniai puslapiai užkraunami daug greičiau.

Toliau detaliau panagrinėsime Ajax komunikavimą JSON formatu, kuris yra alternatyva XML formatui. To tikslas yra parodyti, kad galima kurti dinamiškus puslapius ne vien pasitelkus populiarųjį XML.

2.4. Pažintis su JSON (JavaScript Object Notation) (alternatyva xml formatui)

JSON (JavaScript Object Notation) yra paprastas (lightweight) duomenų apsikeitimo formatas. Tai tektu paremtas, žmonių įskaitomas formatas, skirtas objektams ir kitoms duomenų struktūroms vaizduoti. Pagrindinis JSON tikslas yra perduoti taip struktūrizuotus duomenis tinklu (toks procesas vadinamas serializacija). JSON pagrinde naudojamas Ajax tipo aplikacijose kaip paprasta XML naudojimo alternatyva. JSON yra JavaScript objektų žodinės notacijos poaibis ir paprastai naudojamas su JavaScript, bet kitų kalbų pagrindiniai tipai ir duomenų struktūros taip gali būti atvaizduoti JSON pagalba. Taigi JSON pagalba struktūrizuoti duomenys gali būti perduodami tarp skirtingų programavimo kalbų. Taip pat kodas, skirtas generuoti ir gramatiškai nagrinėti JSON yra galimas daugybėje kalbų. Pvz. C, C++,Perl, PHP ir kitos. JSON naudoja Yahoo, Google ir kitos kompanijos [17].

JSON pagrindiniai duomenų tipai yra :

- Skaičius (sveikas, trukmeninis).
- Eilutė.
- Loginis.
- Rinkinys (array).
- Objektas (rinkinys raktas/reikšmė porų)
- Null [17].

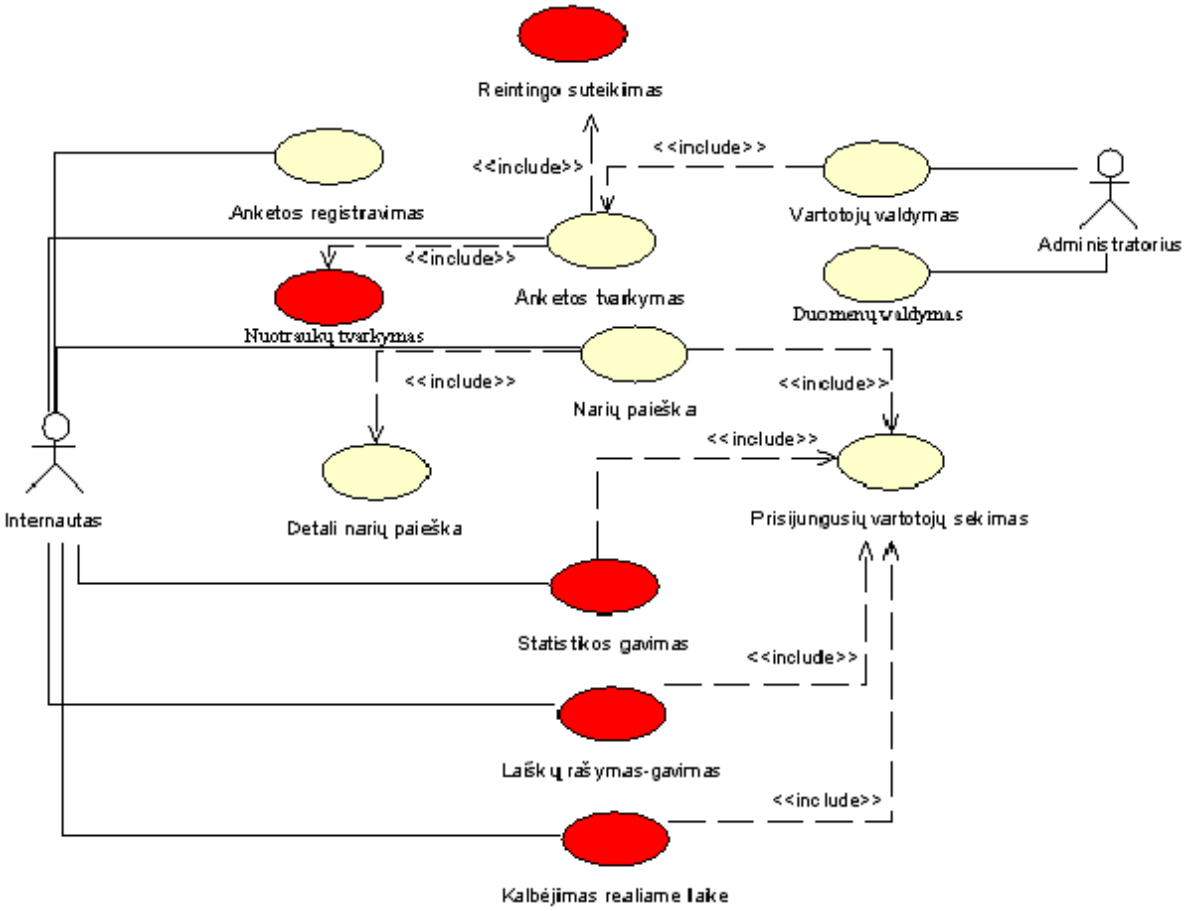
Su Ajax pagalba iš serverio gaunamas su JSON suderinamas atsakymas ir su eval (JavaScript) komanda, tekstinė JSON eilutė perverčiama į objektus [17].

Susipažinę su naudojamomis technologijomis, pasaulinėmis tendencijomis ir būsima ateitimi, pereikime prie konkretesnių modelių, kurie padės atskleisti internetinių puslapių kūrimo su Ajax ypatumus.

3. DINAMIŠKOS SISTEMOS BAZINIO AJAX MODELIO APRAŠYMAS

Pabrėžime, kad sistema kuriama ir modeliai modeliuojami, atsižvelgiant, kad kuriama Java kalba ir technologijomis.

3.1. Panaudojimo atvejai



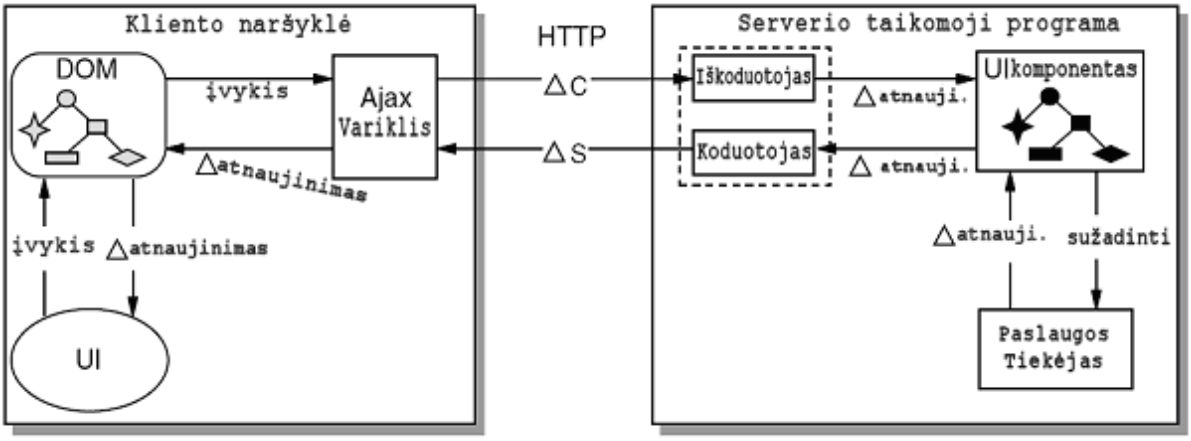
8 pav. Panaudojimo atvejų diagrama

Pirmiausiai apsibrėžime kuriamos sistemos modelio panaudojimo atvejus. Panaudojimo atvejai gali būti, bet kokie, svarbu, kad būtų galima įžvelgti dinamiškumo poreikį juose. Iš panaudojimo atvejų nustatysime kurie panaudojimo atvejai galėtų būti priskiriami prie dinamiškų. Apsibrėžkime dinamišką panaudojimo atvejį. Tai toks atvejis, kuris reikalauja realizacijoje panaudoti Ajax ar kitą technologiją, leidžiančią asinchroniškai kliento naršyklei bendrauti su tarnybine stotimi. Iš 8 paveikslo raudonai pažymėtų ovalų matome, kad dinamiški atvejai bus: „Kalbėjimas realiame laike “, „Reitingo suteikimas “, „Statistikos gavimas “, „Nuotraukų tvarkymas “ ir „Laiškų rašymas-gavimas “ . Žinoma realiai galima visus panaudojimo atvejus priskirti dinamiškiems, bet paminėti atvejai realiausiai atspindi dinamiškų puslapių poreikį ir apskritai prasmę. Dinamiška sistema neįpareigoja visuose puslapiuose naudoti Ajax. Kuriamas modelis apims „ Kalbėjimas realiame laike “ panaudojimo atvejį, kurį laikysime dinamiškiausiau sistemoje. Šis atvejis dar yra ypatingas tuo, kad bus sukurtas

remiantis SPIAR tipo architektūra (detalesnę informaciją apie SPIAR architektūrą galima perskaityti [6] šaltinyje). Kiti dinamiški išrinti atvejai yra palyginus nesudėtingi, todėl juos galima suprojektuoti pasinaudojus jau realizuotais metodais, karkasais, pasinaudojus Tapestry galimybėmis. Tikslas yra parodyti dinamiško puslapio modelį (architektūrą) nuo pat pagrindų, neapvelkant pačių technologijų kitais karkasais, ar palengvinimais, kurie paslėptų pagrindinius Ajax principus. Apie tokį modelį bus kalbama sekančiame skyriuje.

3.2. Dinamiško puslapio architektūra

SPIAR tipo architektūra buvo pasirinkta todėl, kad ji geriausiai parodo Ajax veikimo pagrindus, tuo pačiu aiškiai matomas dinamiško puslapio veikimo principas. Be to, ši architektūra labiausiai tinka dinamiškos internetinės svetainės modelio sukūrimui. Patikslinant – pagrindinio, daugiausiai dinamiškumo reikalaujančiam panaudojimo atvejui, nes SPIAR architektūra tinka daugiau vienam puslapiui, o ne visai sistemai. Bet panaudojus pakartotinį panaudojimą arba Tapestry komponentinio kūrimo galimybę, modelį galime išplėsti visai sistemai, bet kadangi mūsų tikslas yra pademonstruoti pačius Ajax principus, mes nesigilinsime į visos sistemos modelį. 9 paveiksle pavaizduotas SPIAR Ajax architektūros principinė vaizdavimo schema.



9 pav. SPIAR Ajax architektūra [6]

Pagrindiniai SPIAR architektūros elementai yra sudalinti į tris kategorijas: apdorojimo, duomenų ir sujungimo. **Apdorojimo elementai** – tai tokie komponentai, kurie suteikia duomenų elementų transformacijas. Toliau bus aprašomi visi apdorojimo elementai (pavadinimai paryškinti).

Kliento naršyklė palaiko aibę standartų : HTTP, HTML. CSS, JavaScript ir DOM (Document Object Model). Būtent naršyklė pagamina internetinį puslapį atstovaujantį modelį, kad pagaminti vartotojo sąsają.

Ajax variklis jau buvo pristatytas 2.2.1 skyrelyje „Architektūra, veikimo principai “.

Serverio taikomoji programa veikia serveryje ir priima iš tinklo atkeliaujančius HTTP tipo prašymus. Taip pat siunčia atsakymus prašymų siuntėjui.

Paslaugos tiekėjas pateikia serverio logikos variklį, apdoroja būsenų pasikeitimus ir vartotojo pareikalautus veiksmus. Paslaugos tiekėjas gali naudotis, bet kokiais resursais (pvz. duomenų bazė, kitom paslaugom). Tiekėjo funkcionalumas yra sužadinamas įvykių klausytojų, kurie yra prijungti prie komponentų, kurie yra sužadinti įeinančių prašymų.

Koduotojas/iškoduotojas – apdoroja išeinančias/įeinančias žinutes. Koduotojas gavęs žinutę iš kliento naršyklės sugeneruos naują būseną. Iškoduotojas sužadins atitinkamus komponentus serveryje, atliks reikiamus veiksmus ir nusiųs žinutę atgal kliento naršyklei.

UI komponentai - serverio pusėje esantys komponentai, kurie turi savo elgseną ir duomenis.

Duomenų elementai – tokie elementai, kurie turi savyje informaciją, kuri yra naudojama ir transformuojama apdorojimo elementų.

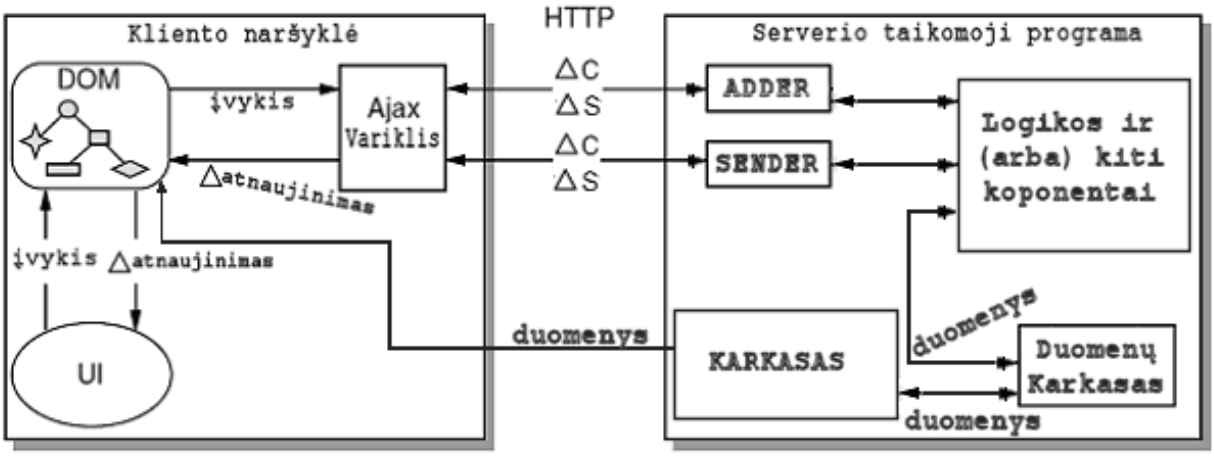
DOM, kuris yra viduje naršyklės, įgavo svarbią rolę Ajax sistemose. Jo pagalba galimi gražūs ir įvairiapusiški efektai. ΔC (duomenys iš kliento) ir ΔS(duomenys iš serverio) skirti duomenų apsikeitimams tarp kliento ir serverio. Duomenų formatas gali variuoti nuo XML iki JSON.

Sujungimo elementai – tai elementai, kurie laiko komponentus kartu, suteikiant jiems galimybę komunikuoti tarpusavyje.

Įvykis yra sužadinamas vartotojo veiksmo, kuris atlieka jį vartotojo sąsajoje. Įvykis patenka į Ajax variklį. Priklausomai nuo įvykio tipo gali būti reikalinga užklausa serveriui arba reikalingas dalinis vartotojo sąsajos atnaujinimas. Įvykis gali būti apdorojamas asinchroniškai. Tai reiškia, kad vartotojo sąsajos kontrolė nedelsiant bus gražinama vartotojui. Serveryje įvykio sukurta užklausa tiesiogiai arba UI komponentų įvykių klausytojų pagalba sužadina kokią nors paslaugą. Δ sujungėjai tai request/response principu per HTTP veikiantis sistemos mechanizmas. Δ atnaujinimai yra naudojami sinchronizuoti serverio ir kliento modeliams.

Pačios architektūros veikimo principas yra nesudėtingas. Vartotojas atlieka veiksmus vartotojo sąsajoje, tie veiksmai sužadina įvykius, kurie Ajax variklio pagalba ir ΔC sujungėjo pagalba nusiunčia užkoduotus duomenis serveriui, kuris iškoduotojo pagalba sužadina reikiamus tų įvykių apdorojimus UI komponentuose. Pvz. duomenų nuskaitymas iš duomenų bazės ir panašiai. Serveris sugeneruos atsakymą ir, užkodavęs su koduotoju, nusiųs klientui pasinaudojęs ΔS. Kliente esantis Ajax variklis nusiųs atnaujinimo komandą DOM, kurio pagalba bus atnaujinta vartotojo sąsaja.

SPIAR architektūrą reikia pritaikyti mūsų kuriamam dinamiškos sistemos modeliui. Daugiausiai pakeitimų bus serverio pusės architektūroje, nes bus stengiamasi suprojektuoti taip, kad kuriama dinamiško puslapio architektūra tarnautų ir kitoms dinamiškos sistemos dalims. Mes koncentruosimės tik į daugiausiai dinamiškumo reikalaujančią sistemos dalį. Kadangi Ajax variklis gali būti bet kokiame kliento naršyklei pasiekiamame sistemos puslapyje – serverio dalis taip pat gali būti pasiekiama nebūtinai tik iš vieno puslapio, turinčio Ajax variklį. Tai labai patogi galimybė, nes galima sukurti visą logiką tuose pačiuose komponentuose, tereikia iškoduoti/užkoduoti naujas žinutes. Apačioje esantis paveikslas parodo abstrakčią dinamiško puslapio architektūrą, kuri realizacijoje bus jau pritaikyta konkrečiai mūsų kuriamai sistemai.



10 pav. Dinamiško puslapio architektūra

Kadangi dauguma puslapių yra kuriami pasinaudojus pagalbinius karkasus – architektūra papildoma duomenų „ Duomenų Karkasas“ ir internetinių sistemų kūrimo karkasais „ KARKASAS“. Duomenų karkaso pavyzdys gali būti Hibernate karkasas, o sistemos: Tapestry, Struts, Spring ir kiti. Logikos ir kiti komponentai gali būti realizuoti pačiame karkase, bet Ajax veikimo principų parodymui logika yra išskirta. „ADDER“ atitinka „ Iškoduotojas“ ir yra skirtas priimti žinutes iš kliento naršyklės. Papildomai pridedama galimybė žinutes apdoroti pačiam ir siųsti atsakymą atgal. Taip padaroma todėl, kad kai kurie veiksmai reikalauja labai mažai logikos ir atsakymas į juos gali būti taip pat labai paprastas. Pavyzdžiui laiško siuntimas. Atsakymai į tokį užklausimą gali būti laiškas išsiųstas arba laiškas neišsiųstas. Bet pagrindinė „ADDER“ paskirtis lieka iškoduoti ateinančias žinutes ir atitinkamai iššaukti veiksmus. „ SENDER“ atitinka „ Koduotojas“. Jo tikslas yra siųsti užkoduotas komandas kliento naršyklei, kad Ajax variklis galėtų atlikti atitinkamus veiksmus vartoto sąsajai. Kliento naršyklė per Ajax kreipiasi į „SENDER“, kad šis jai atsiųstų komandas. Priklausomai nuo logikos, komandų gali prisikaupti ir jas būtina sudėti į eilę. Darome

prielaidą, kad vieno kreipinio į „SENDER“ metu galima gaut tik vieną užkoduotą žinutę iš serverio. Kliento naršyklėje esantis puslapis yra sudarytas iš vartotojo sąsajos elementų, kuriuos galima valdyti DOM ir JavaScript pagalba. Dinamiškame puslapyje turi būti suprojektuota logika, kuri Ajax variklio pagalba būtų susieta su severio pusėje esančia logika. Susiejimas susideda iš galimybės siųsti žinutes serveriui ir galimybės apdoroti atėjusius atsakymus. Taip pat turi būti numatytas kreipinių dažnio reguliavimas pagal veiksmų dažnumą. Galimi du variantai apklausinėjant serverį:

1) Nusiuntus užklausimą į serverį laukti atsakymo, kol jis bus suformuotas ir išsiųstas. Šis požiūris turi problemą. Serveris yra apkraunamas procesais, laukiančiais atsakymų į užklausimą. Taip pat kyla problema, kada iš vis nutraukti laukimą, jei laukimo laikas viršija nustatytas normas. Viskas vykdoma asinchroniškai, todėl vartotojui laukimas nesutrukdys atlikti kitų veiksmų. Tas pats galioja ir 2 variantui. Šis požiūris turi privalumą, kad vartotojas gauna atsaką iš karto, kai jis tik yra suformuojamas.
2) Nusiuntus užklausimą į serverį nelaukti atsakymo. Taip serveryje nebus sukuriama tiek procesų, kiek ateina užklausų. Kliento naršyklė kintančiu dažniu gali apklausinėti serverį, ar nėra naujų komandų. Toks požiūris turi privalumą, kad serveris nebus apkraunamas laukiančiais procesais. Žinoma, yra ir trūkumų. Reikalinga protinga logika, kuri siuntinės užklausimus serveriui. Jei užklausimų siuntimas nebus kontroliuojamas, gali būti perkrautas tinklas užklausomis kaip ir pats serveris.

Mūsų atveju yra pasirenkamas 2 variantas, nes patikrinimas ar yra naujų komandų yra paprasta operacija, nereikalaujanti beveik serverio resursų ir daug lengviau galima patikrinti ar vartotojas dar turi ryšį su serveriu. Taip pat dėl to, kad daroma teorinė prielaida, kad vartotojui bus priimtas galimas rezultato į atliktą komandą užlaikymas. Rastas sprendimas ir užklausų siuntinėjimui. Užklausų siuntimo dažnis bus parenkamas palei vartotojo veiksmų dažnumą. Jei vartotojas neatliks veiksmų, kurie nusiųstu žinutę serveriui, tai ir serveris nebus dažnai apklausinėjamas, dėl naujų komandų . Vartotojui atlikus veiksmą, kurio pasekoje bus išsiųsta žinutė serveriui – bus siunčiamas užklausimas, po nedidelio laiko tarpo, darant prielaidą, kad serveris per tą laiko tarpą jau spės iškoduoti ir apdoroti atėjusią žinutę. Po to bus sumažinamas užklausimų serveriui intervalas, darant prielaidą, kad vartotojas bus atlikęs kitą veiksmą, kuris bus nusiuntęs žinutę serveriui. Jei veiksmas nebuvo atliktas – užklausimų intervalas didinamas iki tam tikros ribos. Taigi tokiu būtų vartotojas maksimaliai turės palaukti pirmos komandos vykdymo rezultato. Pavyzdžiui realiame pokalbyje vartotojas parašęs pirmą žinutę kitam

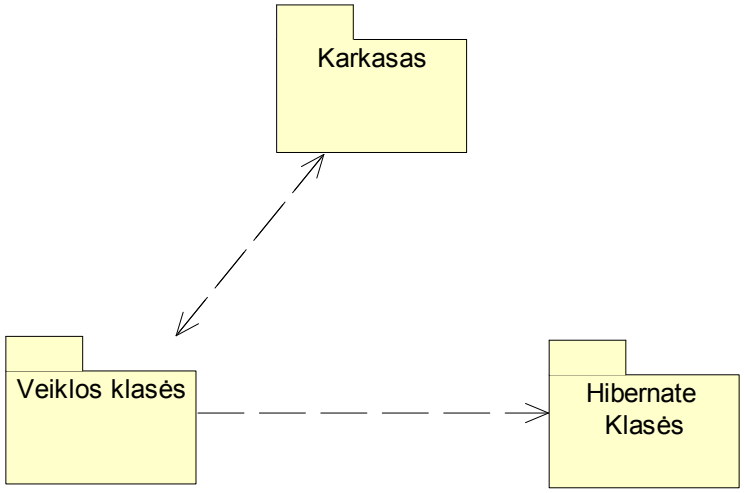
vartotojui, turės palaukt puse sekundės kol bus gautas pranešimas, kad žinutė buvo išsiųsta. Jei per tą laiką jis parašys dar kelias žinutes, jos bus pristatytos akimirksniu. Tokiu principu bus taupomi ne tik serverio bet ir tinklo resursai.

Detalesnę architektūrą padės pateikti dinaminiai ir statiniai sistemos vaizdai, pavaizduti UML diagramų pagalba.

3.3. Klasių diagramos

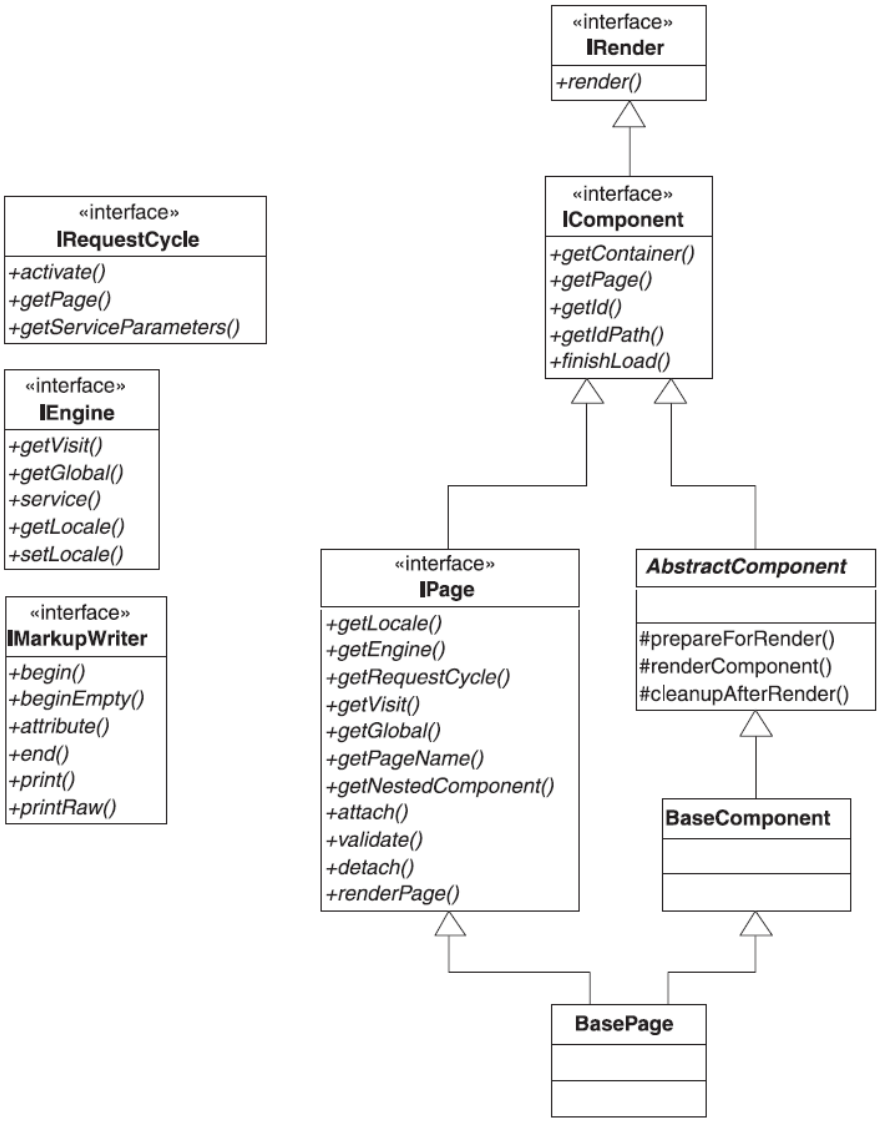
UML diagramų piešimui buvo naudojami Rational Rose ir Omondo Eclipse įrankiai.

3.3.1 Statinis vaizdas



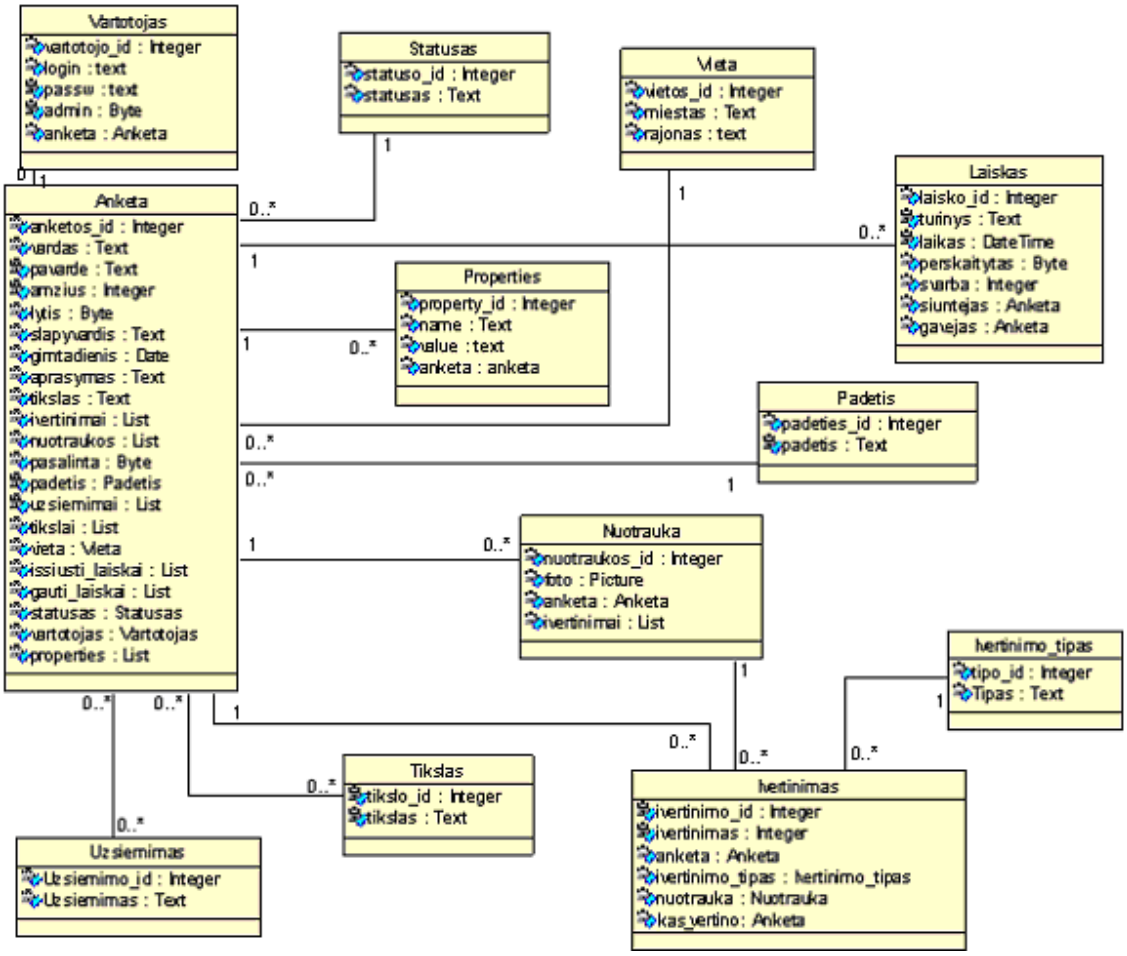
11 pav. Sistemos paketai

Visa sistema susideda iš 3 pagrindinių dalių: duomenų, vaizdavimo ir logikos. Tai standartinis modelis-vaizdas-valdiklis tipo kūrimo šablonas, kurio vienas iš tikslų yra atskirti duomenis nuo vaizdavimo. 11 paveiksle pateiktoje diagramoje matome 3 pagrindinius paketus. „Hibernate klasės“ tai duomenų lygio paketas. Kaip pavyzdys yra įrašyta Hibernate. Vietoj jo gali būti kiti karkasai, skirti darbui su duomenų baze. Hibernate parinktas dėl to, kad yra populiarus ir plačiai naudojamas ir gali dirbti su daugybę duomenų bazių, realiai beveik neperprojektuojant pačios sistemos. „Karkasas“ gali būti, bet koks, bet mes orientuojamės į komponentinius internetinių sistemų kūrimo karkasus, tokius kaip Tapestry. Veiklos klasės realiai gali būti pačio karkaso pakete, bet, atskiriame, kad aiškiau būtų matomos modelis-vaizdas-valdiklis dalys. Kuriame sistema veiks interneto naršyklės pagalba, todėl natūralu, kad vaizdavimas vyks internetinių puslapių pagalbą, kuriais pasirūpins „Karkasas“. 12 paveiksle pavaizduota Tapestry karkaso bazinė klasių diagrama, kurių pagalba bus kuriama vartotojo sąsaja.



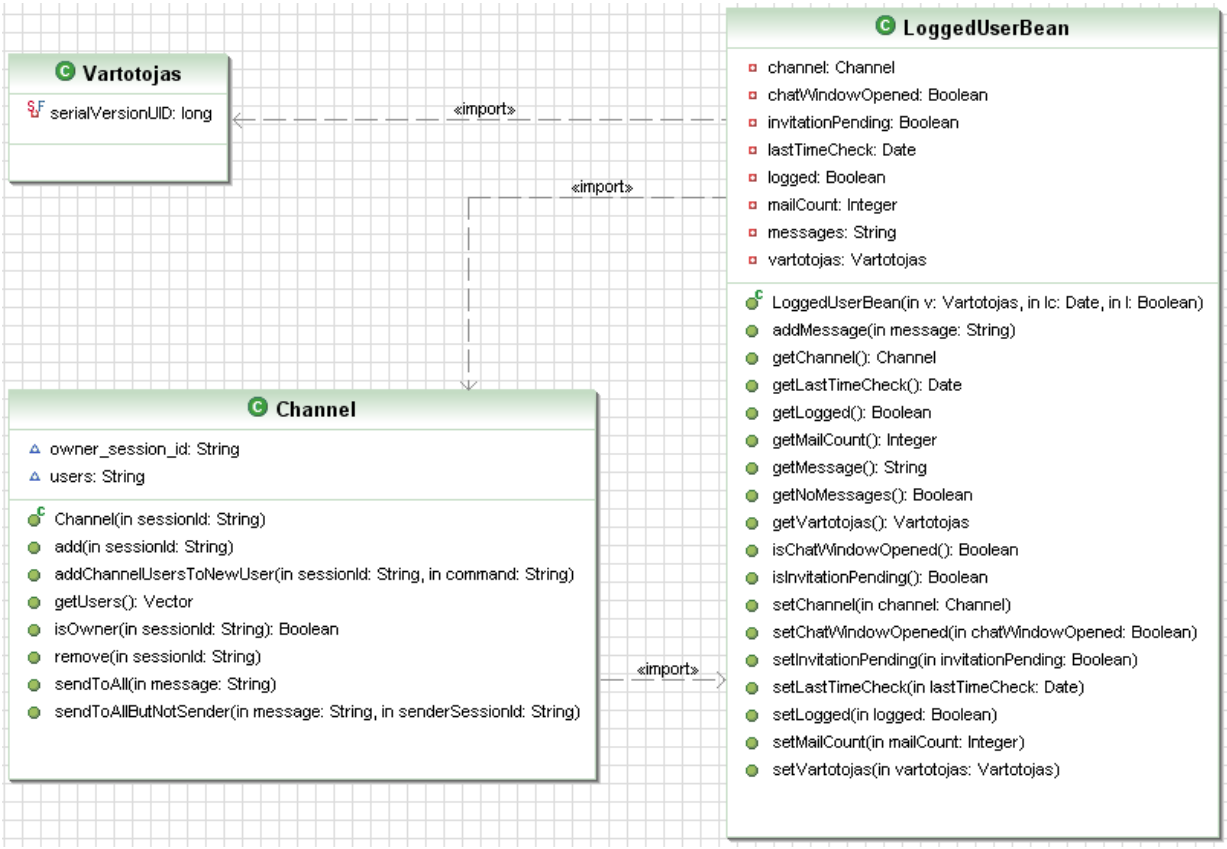
12 pav. Tapestry karkaso sąsajos, klasės ir metodai [18]

Sekančiame 13 paveiksle pavaizduota visos sistemos Hibernate paketo duomenų klasių diagramos 1 dalis. Ji sukurta Hibernate karkasui, norint pritaikyti kitam duomenų valdymo karkasui, reikia pritaikyti klases, to karkaso reikalavimams.



13 pav. Hibernate paketo klasių-duomenų bazės lentelių klasių diagrama

Diagramoje neparodyti taip vadinami „nustatytojai “ ir „ gavėjai “ (angliškai setters ir getters). Pvz. klasėje „Anketa“ yra laukas „vardas “. „Nustatytojas “ bus operacija „setVardas(Text vardas) “ , o „gavėjas “ – „getVardas() “ operacija. Iš diagramos lengva pastebėti, kad kiekviena klasė atitinkamai atitinka viena duomenų bazės lentelę. Mūsų nagrinėjamam dinamiškiausiam panaudos atvejui realiai bus reikalingos tokios, iš 12 paveikslo klasių diagramos, klasės : „Vartotojas “ ir „ Anketa “. 14 paveiksle pavaizduota klasių diagrama parodo ne su duomenų baze susijusias klases ir jų ryšį su 13 paveiksle pavaizduota klasių diagrama.



14 pav. Hibernate paketo ne su duomenų baze susijusių klasių diagrama

14 paveiksle pavaizduota Hibernate paketo klasių diagramos dalis yra pagrindinai skirta mūsų nagrinėjamam „ Kalbėjimas realiame laike “ panaudos atvejui. Kalbėjimui tarpusavyje reikalinga kanalo esybė „Channel “. Kalbėti realiu laiku gali tik prisijungę vartotojai, todėl svarbu turėt prisijungusio vartotojo esybę „LoggedUserBean “, kuri žinotų ne tik, koks vartotojas „Vartotojas “ yra prisijungęs, bet ir galėtų atlikti prisijungusio vartotojo veiksmus. Apibrėšime pagrindinius klasių metodus, kurie daugiausiai įtakos turi mūsų nagrinėjamam panaudos atvejui.

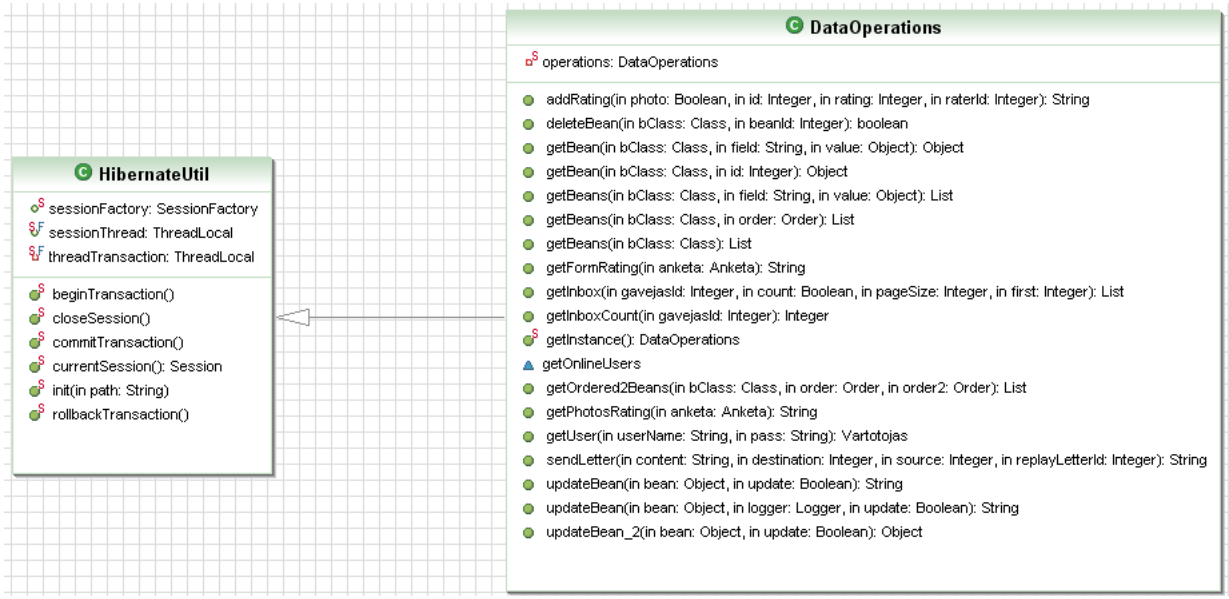
3 lentelė. Klasių aprašymas

Channel	
	public **Channel**(String **sessionId**)
Klasės konstruktorius, skirtas sukurti kanalui, pagal pateiktą sesijos identifikatorių **sessionId.** Taip pat kanalui priskiriamas jo sukūrėjas.	
	public void **add**(String **sessionId**)
Prie kanalo vartotojų sesijų sąrašo pridedama naujo kanalo vartotojo sesija, pagal **sessionId.**	
	public void **addChannelUsersToNewUser**(String **sessionId**,String **command**)
Naujam prisijungusiam vartotojui, kurio sesijos identifikatorius yra **sessionId** nusiunčiami kanalo vartotojai. Nusiuntimo komanda nurodoma **command** parametru.	

	public Vector<String> **getUsers**()
Grąžinamas kanalo vartotojų sesijų identifikatorių sąrašas.	
	public Boolean **isOwner**(String **sessionId**)
Grąžinama ar vartotojas, kurio sesijos identifikatorius **sessionId,** yra kanalo savininkas.	
	public void **remove**(String **sessionId**)
Vartotojas, kurio sesijos identifikatorius yra **sessionId,** pašalinamas iš kanalo.	
	public void **sendToAll**(String **message**)
Visiems kanalo vartotojams nusiunčiama žinutė **message**.	
	public void **sendToAllButNotSender**(String **message**,String **senderSessionId**)
Visiems kanalo vartotojams, išskyrus žinutės siuntėją, kurio sesijos identifikatorius yra **senderSessionId** , nusiunčiama žinutė **message**.	
LoggedUserBean	
	public **LoggedUserBean**(Vartotojas **v**, Date **lc**, Boolean **l**)
Klasės konstruktorius, skirtas sukurti prisijungusį vartotoja, kurio duomenys yra **v.** Nurodoma data **lc**, kada paskutinį kartą vartotojas buvo patikrintas dėl aktyvumo. Taip pat pateikiamas vartotojo prisijungimo požymis **l.** Požymis naudojamas vartotojo prijungimui-atjungimui.	
	public void **addMessage**(String **message**)
Prisijungusiam vartotojui pridedama žinutė **message**.	
	public Channel **getChannel**()
Grąžinamas kanalas, kuriame yra vartotojas.	
	public Date **getLastTimeCheck**()
Grąžinamas laikas, kada vartotojas paskutinį kartą buvo tikrintas dėl veiklumo.	
	public Boolean **getLogged**()
Grąžinamas vartotojo prisijungimo požymis.	
	public Integer **getMailCount**()
Grąžinamas vartotojui atėjusių naujų laiškų skaičius.	
	public String **getMessage**()
Gražinama naujausia žinutė iš vartotojo gautų žinučių sąrašo. Žinutė yra ne tik paprastas tekstas, o kartu ir komanda su parametrais ir tekstu.	
	public Boolean **getNoMessages**()
Grąžinamas požymis ar yra žinučių vartotojo žinučių sąraše.	
	public Boolean **isChatWindowOpened**()
Grąžinamas požymis, ar prisijungęs vartotojas yra atsidaręs pokalbių puslapį.	

	public Boolean **isInvitationPending**()
Grąžinamas požymis, ar vartotojui atėjęs neatsakytas kvietimas kalbėti realiu laiku.	
	public void **setChannel**(Channel **channel**)
Vartotojui priskiriamas kanalas.	
	public void **setMailCount**(Integer **mailCount**)
Nustatomas naujai atėjusių laiškų skaičius **mailCount**.	

15 paveiksle pateikiama duomenų bazės prieigos klasių diagrama.



15 pav. Hibernate paketo duomenų prieigos klasių diagrama

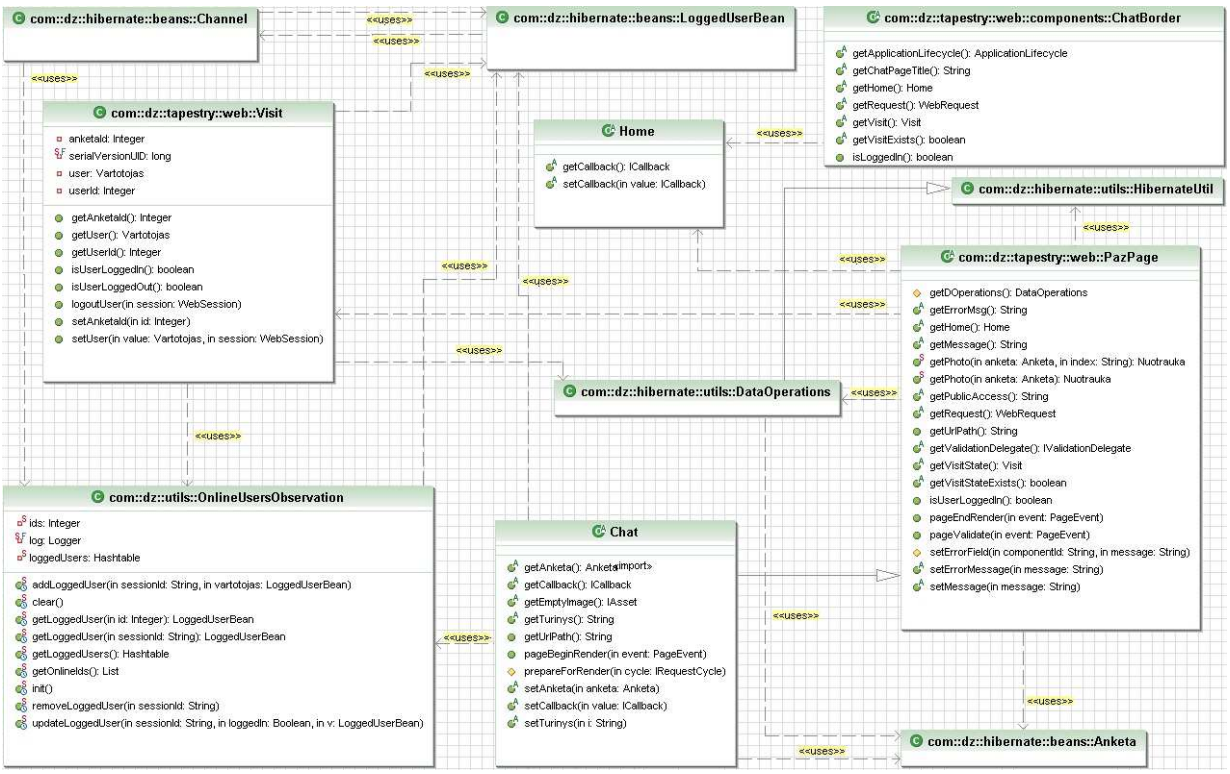
“DataOperations ” klasė naudoja visas 13 paveiksle pavaizduotas klases ir yra vienintele klasė, kurios pagalba atliekami visi veiksmai su duomenų baze. Taigi jei reikėtų šį duomenų bazės klasių modelį pritaikyti kitam duomenų bazės karkasui - reikėtų pritaikyti šią ir kitas Hibernate paketo klases kitai duomenų bazės valdymo sistemai (karkasui), bet nereikėtų keisti visos logikos sistemoje, susijusios su duomenis. Žinoma nedideli pakeitimai būtų reikalingi. Panagrinėsime detaliau kelias klasių funkcijas. Duomenų operacijų, kaip nuskaityti, atnaujinti, įrašyti ar šalinti nekomentuosime. Dinamiškumui parodyti jos neturi įtakos ir neįneša papildomos informacijos.

4 lentelė. Klasių aprašymas

HibernateUtil	
	public static void **init**(String **path**)
Nustatomi Hibernate karkaso pagrindiniai parametrai, kurie nuskaitomi iš **path** nurodytos spartaus nuskaitymo nustatymų bylos. Jei nerandama byla, ji sukuriama ir įrašomi nustatymai	

iš nustatymų pagrindinės bylos. Startuojami visi Hibernate spartinantieji mechanizmai.
public static Session **currentSession**()
Grąžinama aktyvi sesija arba jei jos nėra – sukuriama nauja.
public static void **beginTransaction**()
Startuojama duomenų bazės transakcija.
DataOperations
public String **addRating**(Boolean **photo**,Integer **id**,Integer **rating**,Integer **raterId**)
Įvertinama anketa arba nuotrauka. Parametras **photo** parodo ar bus vertinama nuotrauka, o **id** atitinkamai arba vertinamos nuotraukos arba anketos identifikatorius. Vertinimas yra **rating**, o vertintojo identifikatorius – **raterId**.
public String **getFormRating**(Anketa **anketa**)
Suskaičiuojamas anketos **anketa** įvertinimų vidurkis.
public List **getInbox**(Integer **gavejasId**,Boolean **count**,Integer **pageSize**,Integer **first**)
Grąžinamas vartotojo, kurio anketos identifikatorius yra **gavejasId**, gautų laiškų skaičius. Jei **count** požymis yra **true**, tai grąžinami visi laiškai, jei **false**, tai grąžinama **pageSize** laiškų kiekis, pradedant nuo **first** laiško.
public Integer **getInboxCount**(Integer **gavejasId**)
Grąžinamas vartotojo, kuro anketos identifikatorius yra **gavejasId,** gautų laiškų kiekis.
public List<Anketa> **getOnlineUsers**(List<Integer> **ids**)
Grąžinami prisijungusių prie sistemos vartotojų anketos. Nurodomi prisijungusių vartotojų identifikatorių sąrašas **ids**.
public String **getPhotosRating**(Anketa **anketa**)
Suskaičiuojamas anketos nuotraukų vertinimų vidurkis.
public String **sendLetter**(String **content**,Integer **destination**,Integer **source**,Integer **replayLetterId**)
Nusiunčiamas laiškas, kurio turinys yra **content**. Siuntėjo anketa nurodoma **source** identifikatoriumi, o gavėjo – **destination** identifikatoriumi. Nurodomas (jei ne naujas laiškas) laiško, kuriam bus siunčiamas atsakymas (reply), **replayLetterId** identifikatorius .

Toliau nagrinėsime veiklos logiką, kuri mūsų atveju bus sukurta naudojantis Tapestry karkaso pagalba. 16 paveiksle pateikta sistemos veiklos logikos klasių diagramos dalis, kurioje detaliau nagrinėsime tik klases, konkrečiai susijusias su „Kalbėjimas realiame laike “ panaudos atveju. Visos sistemos logikos klasių diagrama yra labai didelė ir mūsų nedomina, nes mums svarbus tik aukščiau paminėtasis panaudos atvejis ir su juo susijusios klasės.



16 pav. Veiklos logikos klasių diagramos dalis

Sistema kuriama Tapestry karkaso pagrindu, todėl „Visit “, „ChatBorder “, „PazPage “, „Home “ ir „Chat “ klasės yra naudojamos Tapestry karkaso ir yra skirtos būtent sistemos įgyvendinimui šiuo karkasu. Primaname, kad mes nagrinėjame sistemos dalį, o ne visą sistemą. Jeigu norėtume naudoti kitą karkasą, paminėtos klasės, išskyrus „Chat “, nebūtų reikalingos. Apibūdinsime diagramoje pavaizduotas naujas klases. Bus apibūdintos ir tos klasės, kurios beveik bus reikalingos tik Tapestry veikimui, nes taip galėsime susidaryti bendresni galimos dinamiškos sistemos modelio vaizdą prieš pradedant nagrinėti dinamiškos sistemos realizaciją. Jei žiūrėtume vien į projektuojamos sistemos modelį – mus domintu tik „Chat “ klasė. Lentelėse kartu su metodų antraštėmis pateikiamos ir jų anotacijos. Tai Tapestry privalumas, leidžiantis paspartinti kūrimo darbus. Detaliau apie anotacijas galima perskaityti [16] šaltinyje.

5 lentelė. Klasių aprašymas

Home	
Tapestry karkasui reikalinga klasė. Tai paprasto puslapio valdiklio klasė, skirta informacijos rodymui, kai vartotojas nepasirinkęs jokio meniu punkto. Sistemoje gali būti keli namų puslapiai. Pavyzdžiui registruoto/neregistruoto vartotojų namų puslapiai.	
	public abstract ICallback **getCallback**()
Funkcija, kuri grąžina puslapio, iš kurio buvo patekta į namų puslapį, nuorodos objektą.	

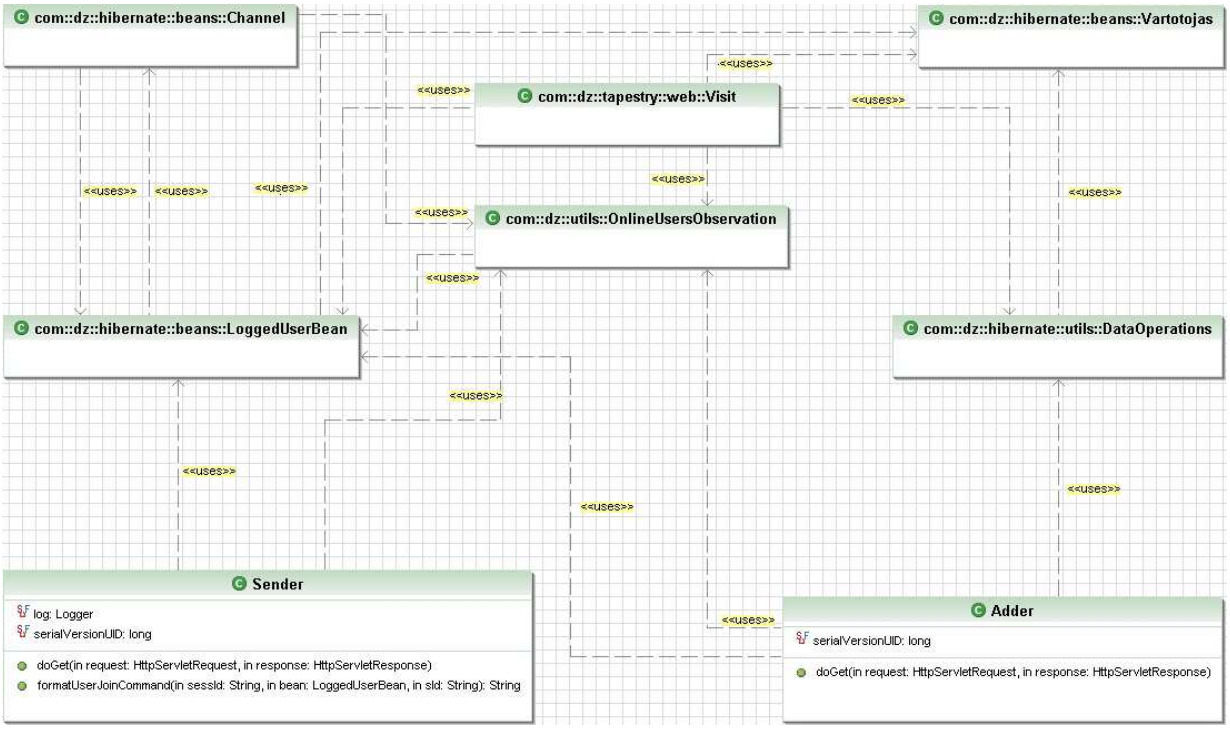
ChatBorder Standartini Tapestry puslapį sudaro turinys ir „rėmelis “ (border), kuriame vaizduojami statiniai sistemos elementai, kaip meniu, pavadinimai ir kita. Puslapio turinio vietoje vaizduojami konkretūs sistemos puslapiai, kaip registraciją, pokalbiai ir t.t. Sistemoje gali būti keli „rėmeliai “. Tokiu būdu galima lengvai keisti pagrindinę sistemos navigaciją ar aplinkos išvaizdą.	
	@InjectPage("Home") public abstract Home **getHome**()
Grąžinamas namų puslapis.	
	@InjectObject("infrastructure:request") public abstract WebRequest **getRequest**()
Grąžinamas užklausimų objektas.	
	@InjectState("visit") public abstract Visit **getVisit**()
Grąžinama vizito klasė. Vizito klasė aprašyta sekančiame klasės aprašyme.	
	@InjectStateFlag("visit") public abstract boolean **getVisitExists**()
Grąžinamas vizito objekto buvimo požymis.	
	public boolean **isLoggedIn**()
Grąžinamas vartotojo prisijungimo prie sistemos požymis.	
	@Message public abstract String **getChatPageTitle**()
Grąžinamas pokalbių realiu laiku puslapio titulinis pavadinimas.	
Visit Vienas iš kertinių, Tapestry karkaso funkcionalumui reikalingų, klasių, joje saugomi duomenys, kurie yra bendri visiems sistemos puslapiams. Pavyzdžiui prisijungusio vartotojo identifikatorius.	
	public Integer **getAnketaId**()
Grąžina prisijungusio vartotojo anketos identifikatorių.	
	public Vartotojas **getUser**()
Grąžinamas prisijungusio vartotojo objektas.	
	public boolean **isUserLoggedIn**()
Grąžinamas vartotojo prisijungimo požymis.	

	public void **logoutUser**(WebSession **session**)
Atjungiamas prisijungęs vartotojas, kurio sesijos objektas yra **session**	
	public void **setUser**(Vartotojas **value**, WebSession **session**)
Prie sistemos prijungiamas vartotojas, kurio objektas yra **value**, o sesijos – **session**.	
PazPage Tai sukurtas tėvinis Tapestry puslapis, naudojamas beveik visų puslapių sistemoje. Jis sutelkia jiems bendrą funkcionalumą vienoje klasėje. Tai pagalbinė klasė, skirta duomenų valdymo reguliavimui, kuris yra reikalingas Hibernate karkasui, taip pat viešų ir apsaugotų puslapių prieigos valdymui.	
	protected DataOperations **getDOperations**()
Grąžinama operacijų su duomenimis klasė.	
	public abstract String **getErrorMsg**()
Grąžinama klaidos žinutė.	
	public abstract String **getMessage**()
Grąžinama pranešimo žinutė.	
	public static Nuotrauka **getPhoto**(Anketa **anketa**,String **index**)
Grąžinamas anketos **anketa** nuotrauka, kurios indeksas yra **index**.	
	public static Nuotrauka **getPhoto**(Anketa **anketa**)
Grąžinama anketos **anketa** pirma nuotrauką. Naudojama prieš tai aprašyta funkcija su **index** parametru lygio vienetui.	
	@InjectMeta("public-access") public abstract String **getPublicAccess**()
Grąžinamas viešo priėjimo prie puslapio požymis.	
	public String **getUrlPath**()
Grąžinamas sistemos internetinis šakninis adresas.	
	@Bean(PazDelegate.class) public abstract IValidationDelegate **getValidationDelegate**();
Grąžinamas sistemos formų laukų pažymėjimų objektas.	
	@InjectState("visit") public abstract Visit **getVisitState**();
Grąžinama vizito klasė.	
	@InjectStateFlag("visit") public abstract boolean **getVisitStateExists**();
Grąžinamas vizito objekto buvimo požymis.	

	public boolean **isUserLoggedIn**()
Grąžinamas vartotojo prisijungimo požymis.	
	@Override public void **pageEndRender**(PageEvent **event**)
Puslapio vaizdavimo baigimo įvykio **event** apdorojimo metodas. Uždaroma Hibernate sesija.	
	public void **pageValidate**(PageEvent **event**)
Puslapio patikros įvykio **event** apdorojimo metodas. Šiame metode patikrinama ar į puslapį galima patekti neprisijungus, jei ne – nukreipiama į namų puslapį.	
	protected void **setErrorField**(String **componentId**, String **message**)
Nustatoma formos lauko **componentId**, kuris netenkina patikros reikalavimų klaidos **message** žinutė.	
OnlineUsersObservation Prisijungusių (ir ne tik) vartotojų sekimo-valdymo klase.	
	public static synchronized void **addLoggedUser**(String **sessionId**,LoggedUserBean **vartotojas**)
Į prisijungusių vartotojų sąrašą įtraukiamas naujas vartotojas **vartotojas**, kurio sesijos identifikatorius yra **sessionId**.	
	public static synchronized void **clear**()
Išvalomas prisijungusių vartotojų sąrašas.	
	public static synchronized LoggedUserBean **getLoggedUser**(Integer **id**)
Grąžinamas prisijungęs vartotojas, kurio identifikatorius yra lygus **id**.	
	public static synchronized LoggedUserBean **getLoggedUser**(String **sessionId**)
Grąžinamas prisijungęs vartotojas, kurio sesijos identifikatorius yra lygus **sessionId**.	
	public static synchronized Hashtable<String, LoggedUserBean> **getLoggedUsers**()
Grąžina visų prisijungusių vartotojų lentelę.	
	public static synchronized List<Integer>**getOnlineIds**()
Grąžinamas prisijungusių vartotojų identifikatorių sąrašas.	
	public static synchronized void **removeLoggedUser**(String **sessionId**)
Pašalinamas prisijungęs vartotojas, kurio sesijos identifikatorius yra lygus **sessionId**.	
	public static synchronized void **updateLoggedUser**(String **sessionId**,Boolean **loggedIn**,LoggedUserBean **v**)
Pakeičiama prisijungusių vartotojų sąraše esančio vartotojo, kurio sesijos identifikatorius yra lygus **sessionId**, informacija **v**. Taip pat nurodomas prisijungimo požymis **loggedIn**.	

Chat
Pokalbių realiu laiku pagrindinis puslapis. Ši klasė gali būti lengvai pritaikoma, bet kokiam kitam java karkasui. Šios klasės html puslapyje: realizuojamas DOM valdymas, Ajax kvietimai ir atsakymų komandų iškodavimas ir vykdymas. Realų puslapį Tapestry sistemoje sudaro: valdiklio klasė, nustatymų xml klasė (nebūtina) ir html puslapis.
public abstract Anketa **getAnketa**()
Grąžinama, bendrautojo realiu laiku, anketa.
@Asset("images/empty.png") public abstract IAsset **getEmptyImage**()
Grąžinamas tuščias paveikslas.
public abstract String **getTurinys**()
Grąžinamas parašytos žinutės turinys.
public void **pageBeginRender**(PageEvent **event**)
Puslapio vaizdavimo pradėjimo įvykio **event** apdorojimo metodas. Nustatoma, kad vartotojas atsidarė pokalbių puslapį.
protected void **prepareForRender**(IRequestCycle **cycle**)
Pasiruošimo vaizduoti puslapį įvykio apdorojimo metodas. Gaunamas ir parametrų ir nustatymų objektas **cycle**.

Liko išnagrinėti 10 paveiksle pavaizduotus „SENDER “ ir „ADDER “ architektūrinius komponentus. Iš esmės, jie yra paprastos java klasės (servlet), turinčios internetinių užklausimų apdorojimui skirtus bazinius metodus. „Adder “ klasė naudojama įdėti vartotojo žinutes (komandas) į vartotojo žinučių eilę. Kai kurias žinutes klasė apdoroja pati, bet daugumą įdeda į eilę „Sender “ klasei apdoroti. „Adder “ klasė pati apdoroja-vykdo pokalbių puslapio lango uždarymo veiksmus, kvietimo sąlygų tikrinimo, paprastų laiškų siuntimą, anketų (nuotraukų) vertinimą ir vertinimų vidurkio perskaičiavimą, vartotojo nuotraukų trynimą ir vartotojui naujai siustų laiškų tikrinimą ir pranešinėjimą. „Sender “ klasė skirta pagrindiniui žinučių apdorojimui, bei naujų žinučių suformavimui ir siuntimui atgal vartotojo naršyklei.



17 pav. Veiklos logikos klasių diagramos dalis (2)

Apibūdinsime naujas klases.

6 lentelė. Klasių aprašymas

Sender	
	public void **doGet**(HttpServletRequest **request**, HttpServletResponse **response**)
Metodas skirtas apdoroti gautoms žinutėms. Atsakymo siuntimo objektas yra **response**. Užklausimo duomenys ir kita atėjusi informacija – **request**.	
	public String **formatUserJoinCommand**(String **sessId**, LoggedUserBean **bean** ,String **sId**)
Grąžina suformuotą naujo vartotojo įtraukimo į kanalą komandą, skirta kliento naršyklei. Vartotojo, kuris kviečia pokalbiams, sesijos identifikatorius yra **sId**, o pakviestojo **sessId**. Kviečiančio vartotojo duomenys – **bean**.	
Adder	
	public void **doGet**(HttpServletRequest **request**, HttpServletResponse **response**)
Metodas skirtas apdoroti kliento žinučių apdorojimui ir naujų formavimui bei siuntimui. Atsakymo siuntimo objektas yra **response**. Užklausimo duomenys ir kita atėjusi informacija – **request**.	

Taigi apibūdinome serverio pusės klases. Liko apibrėžti kliento pusėje esančias funkcijas, skirtas puslapio DOM valdymui arba kitais žodžiais tariant – vartotojo sąsajos pokyčiams atlikti. Be to išliko neapibūdintos funkcijos, reikalingos asinchroniniui bendravimui

su serveriu iš kliento pusės. Kadangi Ajax yra valdomas JavaScript funkcijomis ir „grynas “ JavaScript neturi tiesioginių klasių objektų, lieka apibūdinti funkcijas, kurios gali būti patalpintos į atskirą bylą, arba tiesiog būti „Chat.html “ puslapio viduje. Visam funkcionalumui neužtenka vien tik „Chat.html “, vartotojai kviečiami pokalbiams iš „FormView.html “ puslapio. 7 lentelėje apibrėžiamos pagrindinės JS (JavaScript) funkcijos, reikalingos dinamiškai sistemai (puslapiui).

7 lentelė. Html puslapių ir JavaScript funkcijų aprašymas

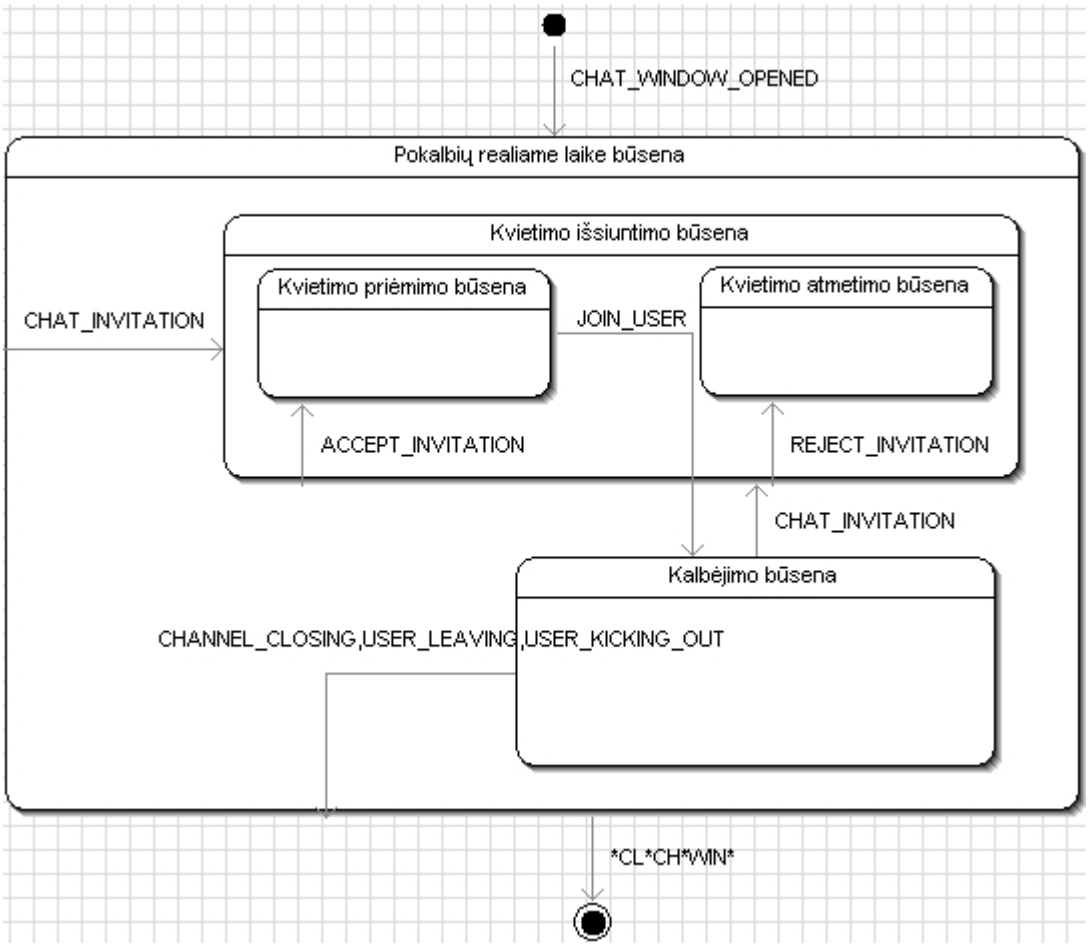
Chat.html
Pokalbių realiu laiku puslapis. Rodomi kanale esantys nariai, bei kanalo žinutės.
function **getAjax**() (funkcija imama iš „**main.js “** bylos)
Naudojama pagrindiniui Ajax valdymo objektui gauti.
function **sendToGetAjaxResponse** ()
Siunčiamas asinhcroninis „SENDER “ komponento (mūsų atveju klasės) sužadinimas. Komponentas, apdorojęs kliento žinutę (jei yra), nusiųs atitinkamas komandas atgal siuntėjui ir suformuos naujas kitiems vartotojams, bei patalpins jas tų vartotojų žinučių eilėse. Taip pat funkcijoje nurodoma funkcija, kuri apdoros gautas komandas iš serverio.
function **sendAjaxMessage**(**message**)
Siunčiama asinchroninė komanda **message** (tekstinio tipo) „ADDER “ komponentui (mūsų atveju klasei).
function **handleAjaxResponse**()
Funkcija, skirta iš serverio gautų komandų ir klaidų apdorojimui. Ši funkcija turi būti nurodoma šios lentelės antroje aprašytoje funkcijoje **sendToGetAjaxResponse**.
function **onWinKeyDown**(**evt**)
Funkcija, apdorojanti puslapio klavišų paspaudimus. Patikrinama ar paspaustas klavišas nebuvo puslapio atnaujinimo tipo.
function **onExitChat**()
Puslapio uždarymo įvykio apdorojimas.
function **processResponseCommand**(**resp**)
Konkrečių komandų **resp** (tekstinis formatas), gautų iš serverio, apdorojimas ir vartotojo sąsajos keitimas.
function **addNewChatUser**(**nick**,**uid**,**aid**,**owner**)
Naujo vartotojo įtraukimas į kanale esančių vartotojų sąrašą ekrane. Nurodomi vartotojo slapyvardis **nick**, identifikatorius **uid**, anketos identifikatorius **aid** ir kanalo savininko požymis **owner**.

	function **userExit**(**id**)
Iš ekrane esančio sąrašo pašalinamas vartotojas, kurio identifikatorius yra **id.**	
	function **channelLost**()
Išvalomas kanale esančių vartotojų sąrašas.	
	function **sendChatMessage**()
Suformuojama ir išsiunčiama pokalbių žinutės siuntimo komanda (kartu su žinutės turiniu).	
	function **setChatMessage**(**user**,**text**)
Į pokalbių langą patalpinama atėjusi tekstinė žinutė **text**. Nurodomas ir žinutės siuntėjas **user**.	
FormView.html Internetinis puslapis, skirtas detaliai vartotojo anketos informacijai pavaizduoti. Taip pat suteikia galimybę suteikti taškus anketai ir (arba) nuotraukoms, leidžia rašyti paprastą laišką ir leidžia kviesti kalbėtis realiu laiku.	
	function **sendAjaxMessage**()
Siunčiamas kvietimas bendrauti realiu laiku.	
	function **getResponseForPage**()
Apdorojamas kvietimo atsakymas.	
	function **trySendMailFromUV**()
Bandoma išsiųsti paprastą laišką.	
	function **mailSendResponse**()
Apdorojamas laiško siuntimo atsakymas.	
	function **sendPhotoRating**(**photoId**,**rating**,**rater**)
Siunčiamas nuotraukos, kurios identifikatorius yra **photoId,** įvertinimas **rating**, nurodant vertintojo identifikatorių **rater**.	
	function **renewPhotoRating**()
Atnaujinamas bendras anketos nuotraukų vertinimų vidurkis.	
	function **sendFormRating**(**formId**,**rating**,**rater**)
Siunčiamas anketos, kurios identifikatorius yra **formId,** įvertinimas **rating**, nurodant vertintojo identifikatorių **rater**.	
	function **renewFormRating**()
Atnaujinamas bendras anketos vertinimų vidurkis.	

Taigi išnagrinėjome dinamiškos sistemos statinį vaizdą. Pamatėme kokios klasės yra reikalingos, kokios būtinos ir pan. Dabar reikia nustatyti kaip sistema veikia, kokios jos būsenos ir kaip veikia dinaminiai mechanizmai.

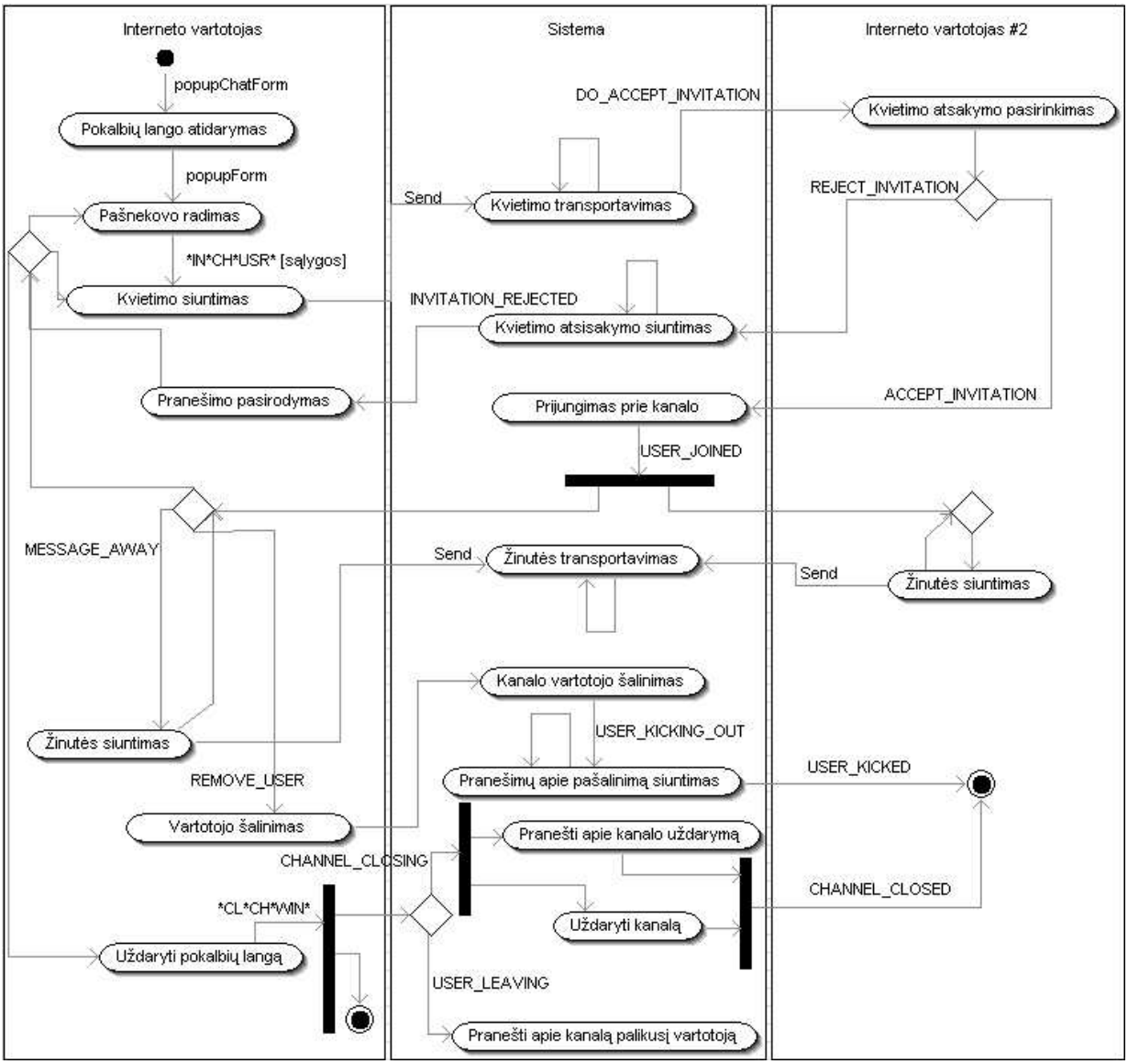
3.3.2 Dinaminis vaizdas

Pirmiausiai aptarsime būsenas, kurios galimos, bendraujant realiu laiku.



18 pav. Pokalbių realiu laiku būsenų diagrama

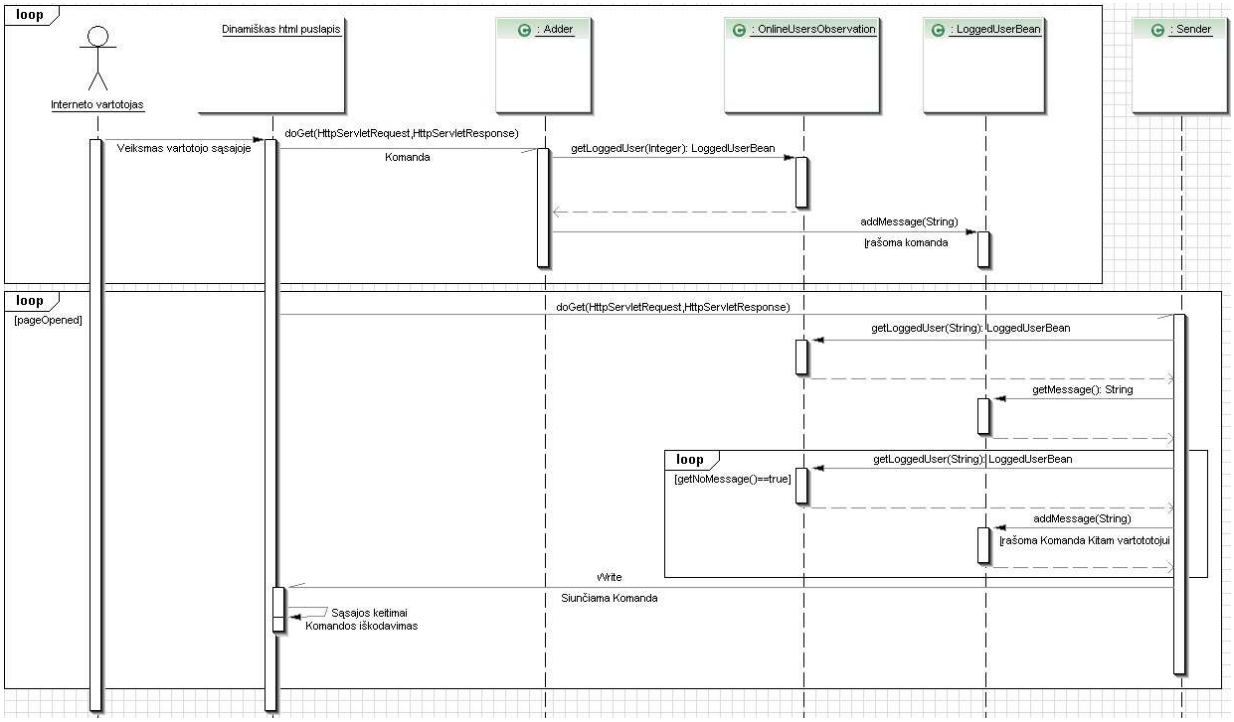
Kaip matome 18 paveiksle – visi veiksmai, susiję su bendravimu realiu laiku yra galimi tik „Pokalbiai realiame laike “ būsenoje, į kurią patenkama atsidarius pokalbių puslapį. Sekančiame paveiksle esanti veiklų diagrama parodo vartotojų veiklas, kurios gali vykti atsidarius pokalbių langą.



19 pav. Pokalbių realiu laiku veiklos diagrama

Vartotojas atsidaręs pokalbų puslapį gali kviesti pokalbiams kitą vartotoją, jei pašnekovas tenkina tokias **sąlygas** (sąlygos) : vartotojas yra prisijungęs, neturi sukūręs savo kanalo, nepakviestas kito vartotojo ir yra atsidaręs pokalbių langą. Kvietimas kalbėti vykdomas išsiunčiant kvietimą kitam prisijungusiam vartotojui. Toliau reikia sulaukti kviečiančiojo atsakymo. Kviečiamas vartotojas gali priimti arba atmesti kvietimą. Jei jis priima kvietimą, tada patenka į kviečiančiojo kanalą ir tada jau gali kalbėtis su juo. Jau vykstant pokalbiui vartotojas gali pasikviesti kitus sistemos naudojus prisijungti prie kanalo. Pokalbis baigiamas, kai : kanalo savininkas uždaro kanalą, išmeta vartotoją arba pats vartotojas (ne kanalo savininkas) palieka sistemą. 19 paveiksle pavaizduota pagrindinė vartotojų veiklos diagrama. Puslapio uždarymo įvykis „CL*CH*WIN* “ iškviečia sistemos reakciją. Likusiam (-iems) kanalo vartotojoms pranešama apie kanalą palikusi vartotoją. Jei palieka kanalo savininkas, visi vartotojai nebegali toliau bendrauti. Kanalo savininkas gali išmesti

(REMOVE_USER) vartotoją iš kanalo. Realiai sistema yra sudėtingesnė, nei pavaizduota 19 paveiksle. Detaliau panagrinėsime sudėtingesnes veiklas. Tam pasitelksime sekų diagramas. Pradėsime nuo „Kvietimo siuntimas “ veiklos. Kaip jau buvo minėta, kviesti kalbėtis galima tik tada, jei įvykdomos tam tikros sąlygos. 20 paveiksle pavaizduota bendra dinamiškos sistemos veikimo sekų diagrama, kai „Adder “ klasė pati neapdoroja jokių žinučių. Darome prielaidą, kad visos žinutės pasiekia serverį akimirksnių, be jokio užlaikymo.

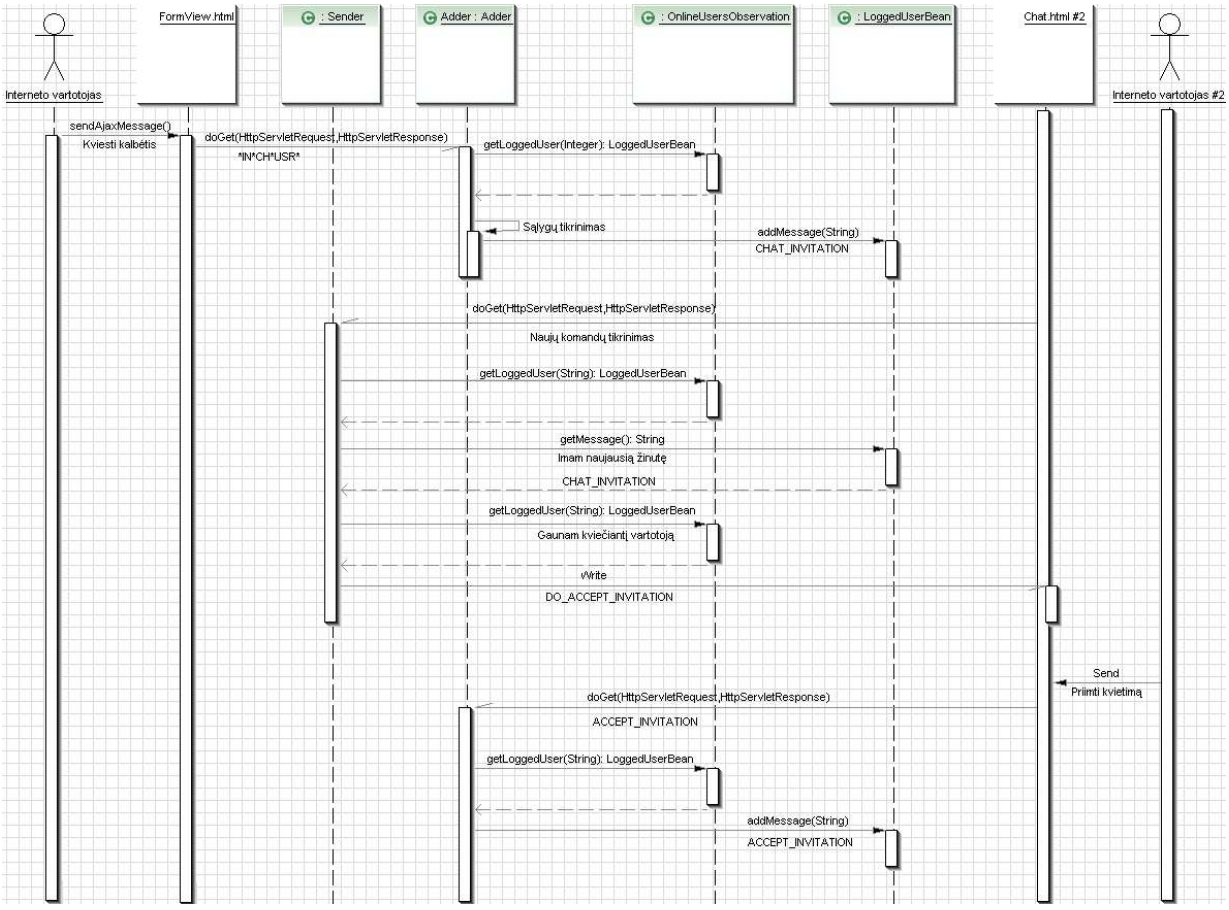


20 pav. Dinamiškos sistemos veikimo sekų diagrama

Kaip galima matyti 20 paveiksle, sistema veikia pilno duplekso principu. T.y. tarp serverio ir kliento sudaromas abipusis ryšys. Naudojami skirtingi Ajax varikliai žinutėms nusiųsti ir gauti. Interneto vartotojas gali atlikti tam tikrus veiksmus sistemos vartotojo sąsajoje, kurie reikalauja sąveikos su serveriu. Ar tai duomenų atnaujinimas, įrašymas ir panašiai. Norėdami sumažinti sudėtingumą sekų diagramose, nebevaizduosime „Sender “ klasėje esančio veiksmo, kuris kartojasi, visais atvejais, naudojant šią klasę. Tai prisijungusio vartotojo (LoggedUserBean) gavimas, kuris siunčia komandą „Sender “ klasei.(21 paveiksle dar bus naudojamas). 20 paveiksle esančioje sekų diagramoje nagrinėjami tik tie veiksmai, kurie reikalauja sąveikos su serveriu. Arba kitais žodžiais tariant – reikalauja asinchroninio kreipimosi į serverį. Kai vartotojas atlieka veiksmą, Ajax variklis nusiunčia užkoduotą komandą į serverį, kur žinutę iškoduoja „Adder “ klasė ir įrašo ją į vartotojo žinučių eilę. Tai gali kartotis, atitinkamai kiek veiksmų atliks vartotojas. Lygiagrečiai puslapis apklausinėja serverį, „Sender “ klasė paima vieną naujausią žinutę, iš apklausiančio vartotojo žinučių eilės

ir ją apdoroja. Apdorojimo metu suformuoja naujas žinutes kitiems vartotojams. Tai gali kartotis, kol nebebus naujų žinučių. Žinučių skaičius priklauso nuo veiksmo ir nuo kanale esančių vartotojų skaičių, jei formuojama ir siunčiama masinė žinutė. Serveris yra apklausinėjamas kintančiu dažniu. Dažnis mažėja, jei vartotojas neatlieka jokių veiksmų, kurie įneštų naujų žinučių. Dažnis padidėja iki maksimalaus, atsiradus, bet kokiai komandai, skirtai vartotojo sąsajai. Jei pokalbių puslapis uždaromas – serverio apklausinėjimas sustabdomas. Toliau nagrinėsime pagrindines veiklas, skirtas bendravimui realiu laiku. Pradėsime nuo kvietimo kalbėti siuntimo

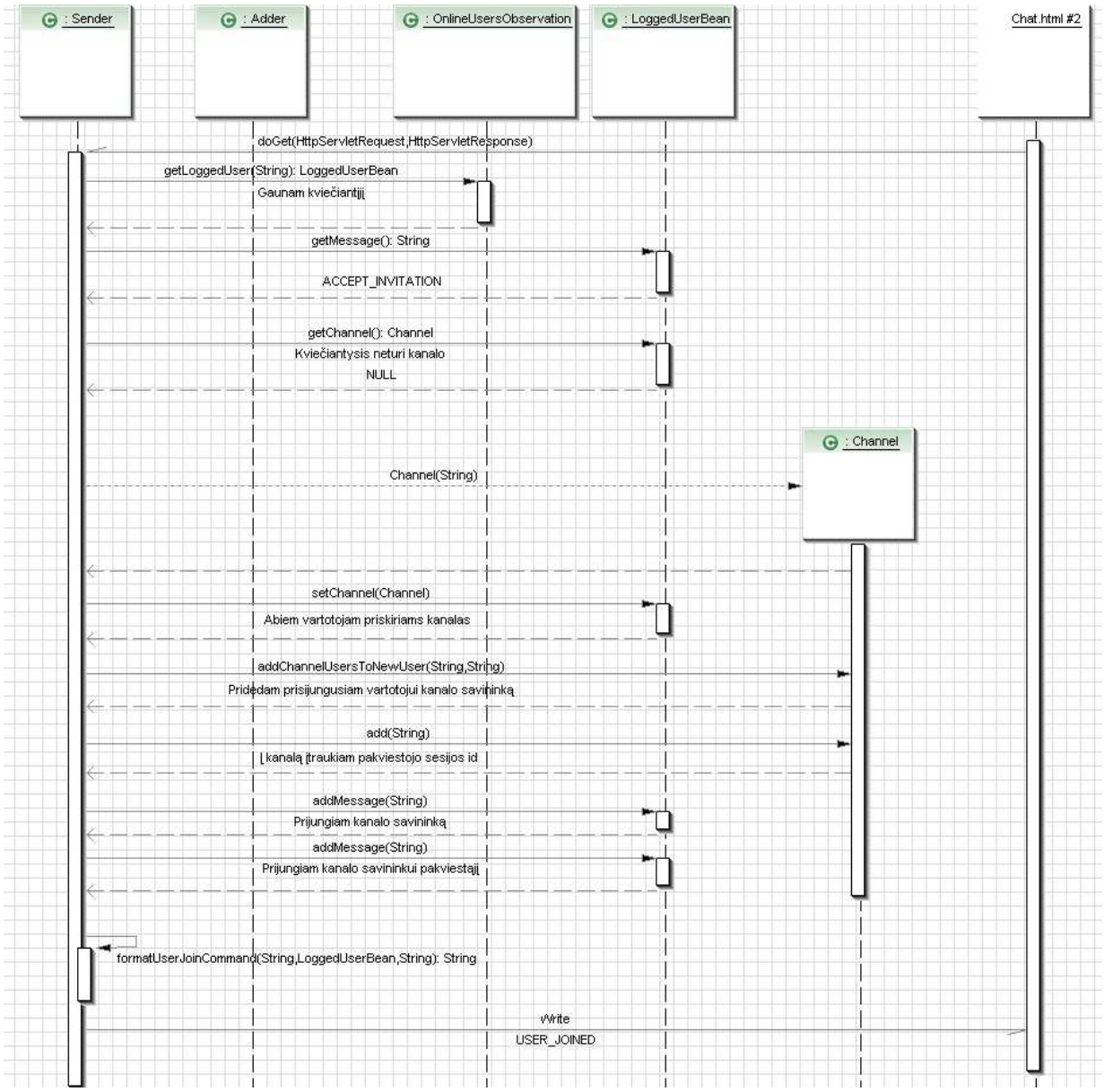
Kvietimas sukuriamas kai „Adder " klasė gauna komandą „*IN*CH*USR* ". „Adder " klasė patikrina kvietimo siuntimo sąlygas ir jei jos yra tenkinamos - kvietimo komanda yra įrašoma į adresato žinučių eilę. Kviečiantysis gauna kvietimą, kuri gali priimti arba atmesti. Tarkime, kad jis priima kvietimą. Šis procesas yra pavaizduotas 21 paveiksle esančioje sekų diagramoje.



21 pav. Kvietimo priėmimo sekų diagrama

Kai kviečiantysis patvirtina kvietimą – prasideda kitas etapas, kurio metu yra sukuriamas kanalas (jei dar nesukurtas nei pas kviečiamą, nei pas kviečiantį) ir kviečiantysis ir kviečiamasis yra prijungiami prie sukurto kanalo, kviečiantį priskiriant kanalo savininku.

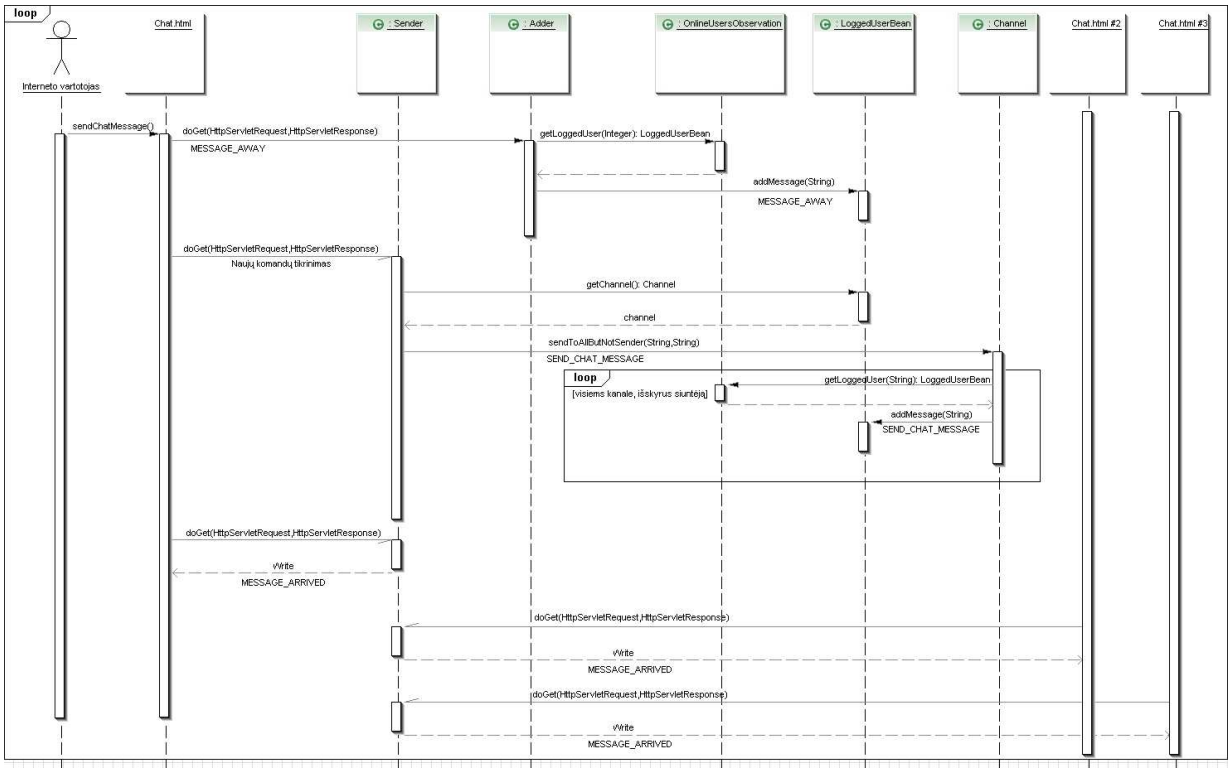
Kūrimas vyksta iš kviečiančiojo pusės, nes jis patvirtino kvietimą ir jo žinutė pirma nukeliauja į jo komandų eilę. Realiai nėra jokio skirtumo, kad inicijuoja kanalo kūrimą. Kanalo savininkas gali uždaryti kanalą arba išmesti kitus pakviestus vartotojus lauk iš kanalo. Tai tik viena veiksmų alternatyva. 22 paveiksle pavaizduota kanalo sukūrimo ir naujo vartotojo įtraukimo į kanalą sekų diagrama. Tai, galima sakyti, 21 paveiksle pavaizduotos sekų diagramos veiksmų pratęsimas.



22 pav. Kanalo sukūrimo ir vartotojų prijungimo sekų diagrama

Kviečiamam vartotojui priėmus kvietimą, jo žinučių eilėje įrašoma kvietimo priėmimo žinutė. Kai ją pradeda apdoroti „Sender “ klasė : sukuriamas kanalas ir priskiriamas tiek pakviestajam, tiek kviečiančiajam. „addChannelUsersToNewUser “ metodas įrašo naujai pakviestam vartotojui esamus kanalo vartotojus. Įrašymas vyksta pridedant prijungimo

komandą į pakviestojo žinučių eilę. Kadangi kanalas buvo naujau sukurtas, tai pridedamas tik kanalo savininkas. Kanale įrašomas pakviestojo sesijos identifikatorius, kad būtų galima atlikti kanalo veiksmus. Galiausiai prijungiamas ir pats kanalo savininkas, įrašant prijungimo komandą į jo žinučių eilę. Taip pat jam prijungiamas ir naujai pakviestas vartotojas. Prijungimą dar paprastintai galima įvardinti, kaip vartotojo sąsajoje esantį kanalo sąrašo atnaujinimą nauju vartotoju. Kadangi operacijos inicijuotos iš pakviestojo pusės, telieka suformuoti prijungimo komandą ir nusiųsti jau tiesiai pakviestojo vartotojo naršyklei. Egzistuoja ir kitos situacijos. Pavyzdžiui naujo vartotojo prijungimas prie jau esamo sukurto kanalo ir vykstančio pokalbio. Veiksmai realiai yra identiški, išskyrus tai, kad nereikia kurti naujo kanalo. Prie esamo kanalo prijungiamas naujas vartotojas, jam prijungiami esami kanalo vartotojai, ir kiekvienam kanalo vartotojui prijungiamas pakviestasis. Kadangi nagrinėjame „ Kalbėjimas realiame laike “ panaudos atvejį, belieka tik pavaizduoti patį kalbėjimo procesą. Darome prielaidas, kad kanalas sukurtas, kanalo savininkas yra pakvietęs du naujus vartotojus. 23 paveiksle pavaizduota pokalbių realiu laiku sekų diagrama.



23 pav. Pokalbių realiu laiku sekų diagrama

Bet kuris kanalo vartotojas gali išsiųsti žinutę. Žinutės išsiuntimo į kanalą komandą patalpinama siuntėjo komandų eilėje. „Sender “ klasė apdoroja šią žinutę. Siunčiama pokalbių žinutė visiems kanale esantiems vartotojams. Siuntėjas gauna žinutę pirmas, nes jo naršyklės

užklausimas dabar yra vykdomas. Galiausiai visi kanalo dalyviai, atitinkamais laiko intervalais, gauna išsiųstą žinutę.

Taigi išanalizavome statinį ir dinaminį sistemos vaizdus. Nusistatėme dinamiško puslapio architektūrą ir pačios sistemos veikimo principus. Iš bendresnio apibrėžimo pereikime prie konkrečios eksperimentinės sistemos, skirtos pademonstruoti dinamišką sistemą ir jos privalumus. Nors modelis buvo kuriamas remiantis Tapestry karkasu, jo realizacija ir pats veikimo principas yra bendras daugumai kitų java karkasų.

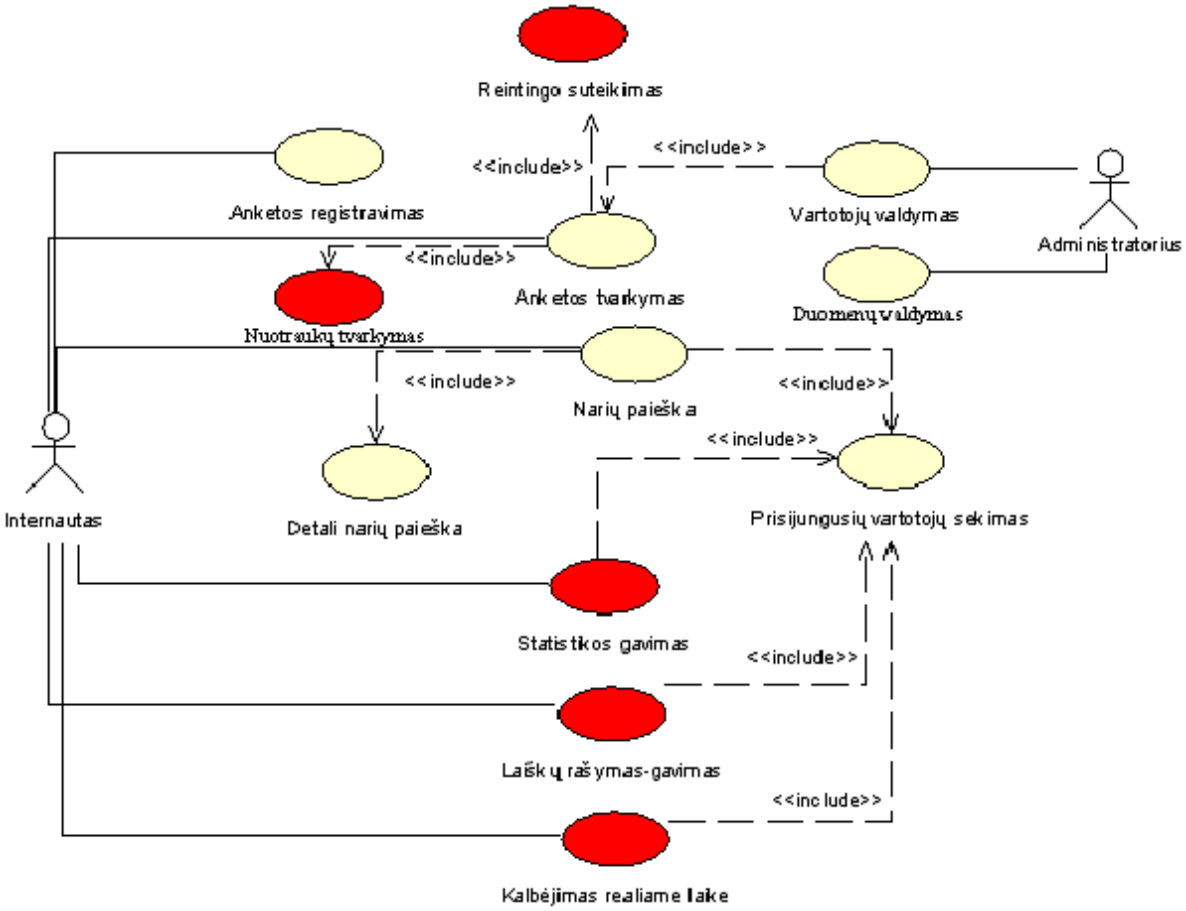
4. SISTEMOS PROGRAMINĖ REALIZACIJA IR DINAMIŠKŲ PUSLAPIŲ SU AJAX TYRIMAS

4.1 Eksperimentinės sistemos aprašymas

Eksperimentinė sistema pademonstruoja apibrėžtą dinamiškos sistemos modelį jo konkrečioje realizacijoje. Kad sistema nebūtų vien tik parodomi – buvo pridėta daugiau sistemos funkcijų, kurios nėra skirtas demonstruoti dinamiškai sistemai, bet yra, kad sistema galėtų veikti kaip visuma. Taigi buvo sukurta pažinčių svetainė, skirta bendrauti internetu. Sistemoje numatyti du vartotojų tipai: administratorius ir sistemos naudotojas. Sistemą geriausiai apibūdina jos atliekamos funkcijos. Sistema sukurta naudojant Tapestry ir Hibernate karkasus (neįskaitomos kitos reikalingos bibliotekos, paminėtų karkasų pilnam veikimui). Sistema sukurta „Internet Explorer 6.0“ ir daugiau naršyklei.

4.1.1 Sistemos funkcijos

Sistemos funkcijas geriausiai atspinti jos panaudojimo atvejų diagrama, pavaizduota 24 paveiksle. „Kalbėjimas realiame laike “ jau buvo aptartas nagrinėjant dinamiškos sistemos architektūrą. Prie dinamiškų panaudojimo atvejų priskyrėme: „Kalbėjimas realiame laike “, „Reitingo suteikimas “, „Statistikos gavimas “ ir „Laiškų rašymas-gavimas “. Paminėti panaudos atvejai daugiausiai reikalauja dinamiškumo, todėl pagrindinai orientuosimės į juos apibrėžiančias funkcijas. Mūsų nedomina tokie procesai, kaip prisijungimas, registracija ar paieška.



24 pav. Sistemos panaudojimo atvejų diagrama

Pagrindinės sistemos funkcijos:

A. Registracija, anketos informacijos keitimas.
B. Nuotraukų įkėlimas ir šalinimas (naudojamas Ajax).
C. Bendravimas realiu laiku, siunčiant tekstines žinutes (naudojamas Ajax).
D. Laiškų rašymas ir siuntimas kitiems vartotojams (naudojamas Ajax).
E. Rankinis ir automatinis gautų laiškų tikrinimai (automatiniam naudojamas Ajax)
F. Prisijungusių ir apskirtai narių paieška.
G. Įvertinimo suteikimas vartotojo anketai ir (arba) jos nuotraukoms (naudojamas Ajax).
H. Statistikos apie gautus įvertinimus pateikimas (naudojamas Ajax).
I. Vartotojų valdymas.
J. Klasifikatorių duomenų tvarkymas (naudojamas Ajax).

Funkcijų sąraše pabrauktas svarbiausia, sudėtingiausia ir daugiausiai dinamiškumo reikalaujanti funkcija. Kaip matyti iš sąrašo, beveik visos funkcijos naudoja Ajax. Naudojimo lygis neapibrėžiamas. Vienoms funkcijoms jo reikia pagrindinai tik duomenų apsikeitimui su serveriu. Tai D, E ir J funkcijos. Kitoms funkcijoms neužtenka tik duomenų apsikeitimo. Tai likusios B, C, G ir H. Šias likusias funkcijas dar galime išskaidyti į dvi dalis. Pirmoje būtų B,

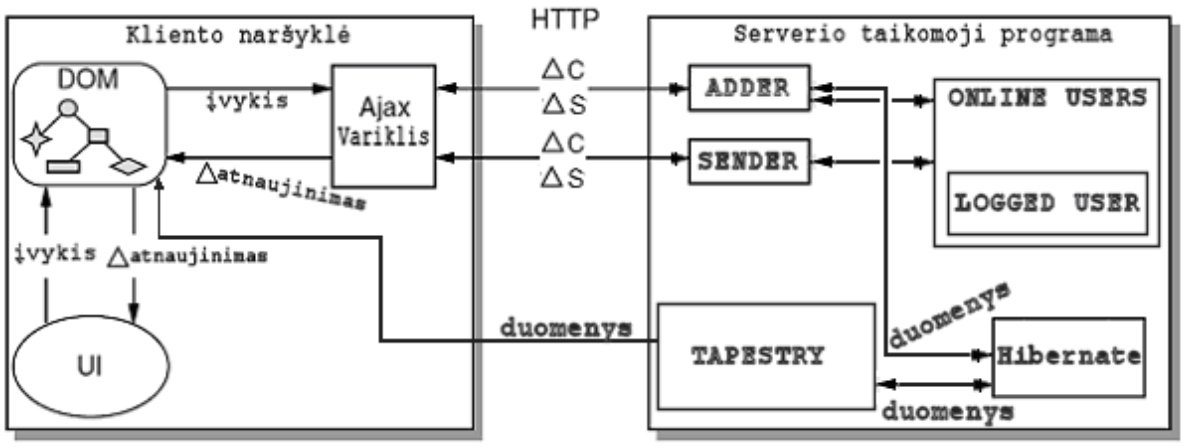
G ir H, o antroje – C. Išskaidėme atsižvelgdami į vartotojo veiksmų ir galimybių skaičių. B, G ir H funkcijos nereikalauja daug vartotojo veiksmų. Priešingai nei C funkcija, kuri po savimi slepia dar tokias vidines funkcijas:

A. Žinutės rašymas ir siuntimas į kanalą.
B. Kanalo vartotojo šalinimas.
C. Vartotojo kvietimas bendrauti.
D. Žinutės gavimas.
E. Kanalo uždarymas.
F. Kvietimo kalbėti priėmimas-atsisakymas.

Apibrėžę funkcijas, pereikime prie konkrečių jų realizacijų sukurtoje eksperimentinėje sistemoje.

4.1.2 Dinamiškų sistemų (puslapių) realizacija su Ajax ir Tapestry

Iš pradžių patikslinsime 10 paveiksle pavaizduotą dinamiškos sistemos architektūra, pritaikydami ją konkrečiai mūsų eksperimentinei sistemai.

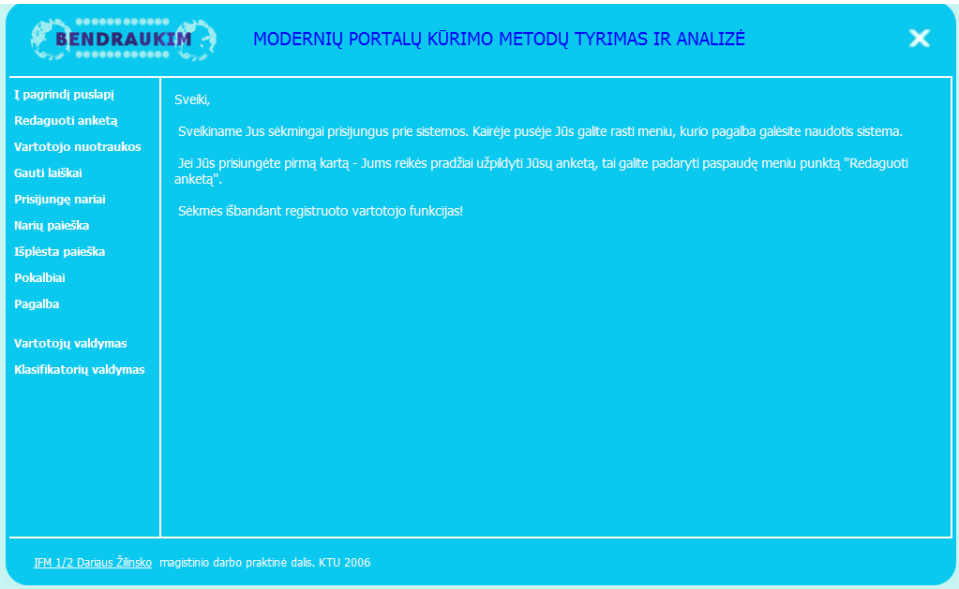


25 pav. Eksperimentinės sistemos architektūra

Sistema sukurta naudojant viską jungiantį Tapestry karkasą. Duomenų bazės prieigos ir valdymui buvo naudojamas Hibernate karkasas. Funkcijų veiksmams įgyvendinti naudojami papildomi prisijungusių vartotojų komponentai (ONLINE USERS). Jie reikalingai pagrindinai pokalbiams realiu laiku. Prisijungęs vartotojas gali turėti savo kanalą arba prisijungti prie kito vartotojo sukurto kanalo. Pats veikimo principas išlieka toks pat. 25 paveiksle pavaizduota daugiau C funkcijos veikimo architektūra. Taip yra todėl, kad mes šią funkciją laikome labiausiai atspindinčia dinamišką sistemą. Kitos funkcijos nereikalauja tokių dažno asinchrnoninio bendravimo su serveriu. Taip pat ši funkcija nenaudoja jokių kitų karkasų

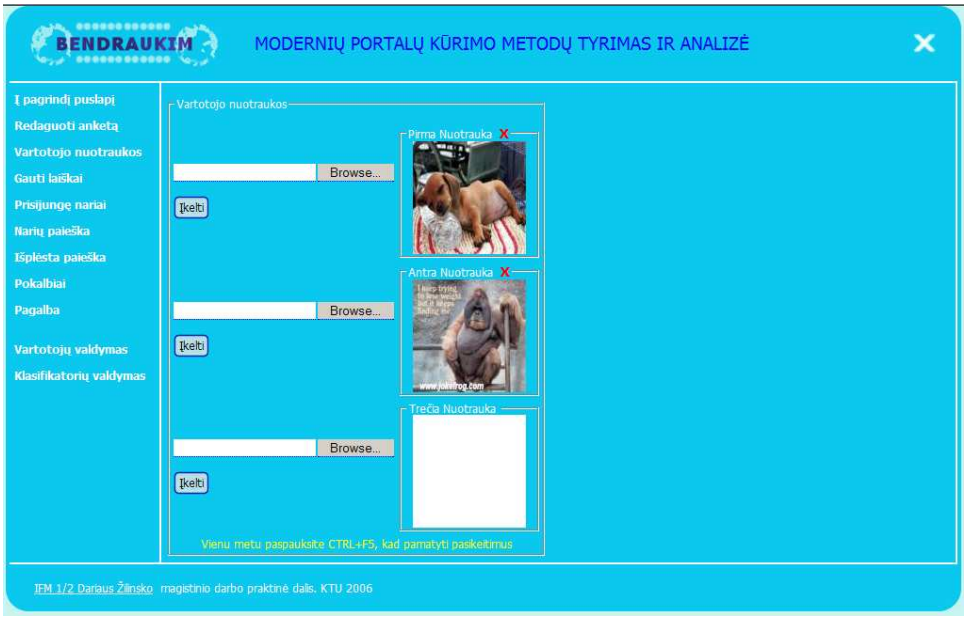
apvalkalų, kurie paprastina Ajax naudojimą. To tikslas yra pademonstruoti pačius Ajax veikimo principus.

Pradėsime nuo paprastesnių sistemos dalių. Nenagrinėsime registracijos, prisijungimo ir anketos redagavimo funkcijų, nes jos neteikia naudingos informacijos, susijusios su dinamiškais puslapiais (sistemomis). Sistema, kaip ir dauguma internetinių svetainių, prisijungusį vartotoją pasitinka **pagrindiniu puslapiu**, kuriame jis gali atlikti visas jam, sistemos teikiamas, funkcijas.



25 pav. Registruoto vartotojo pagrindinis puslapis.

Pirmas susidūrimas su Ajax vartotoją pasitinka anketos **nuotraukų įkėlimo** puslapyje.



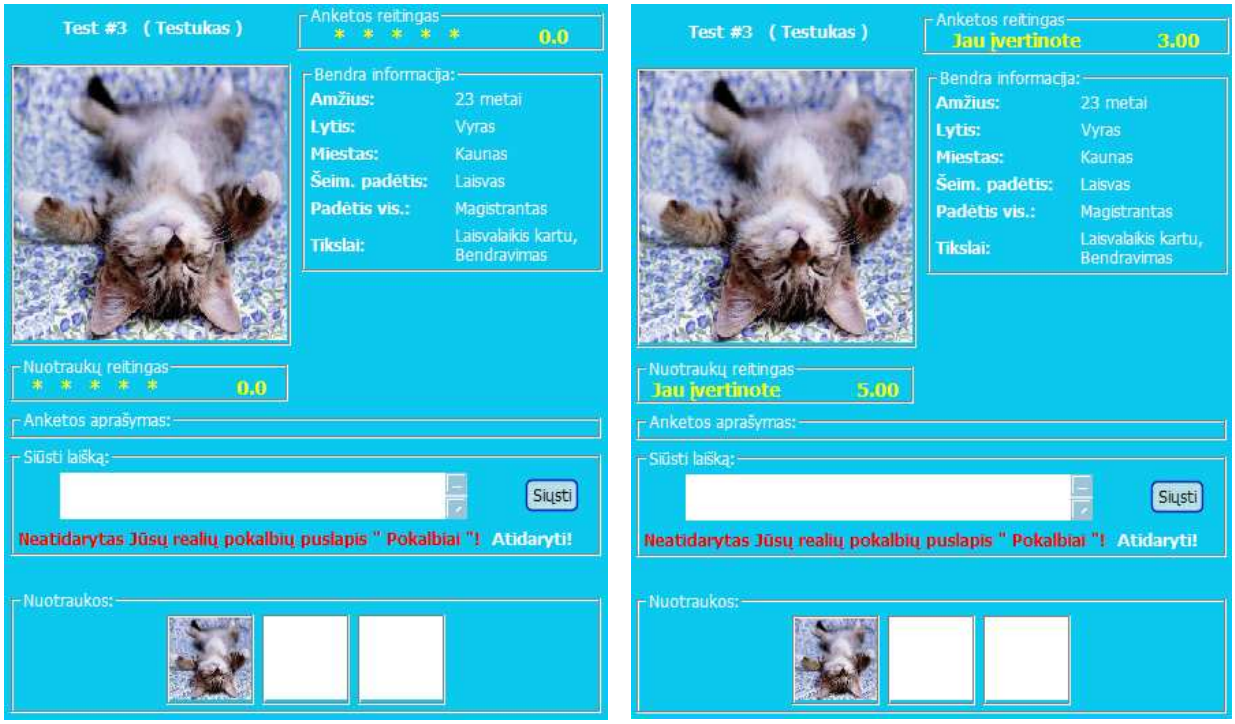
26 pav. Anketos nuotraukų įkėlimo puslapis

26 paveiksle pavaizduote puslapyje Ajax yra naudojamas sėkmingo pasirinktos nuotraukos ištrynimui pranešti. Sekantis žingsnis užpildžius anketos informaciją – susirasti

pašnekovų. Tarkim ieškome pašnekovų realaus laiko pokalbiams. Tam mums reikės **prisijungusių vartotojų paieškos** puslapio, kuris pavaizduotas 27 paveiksle. Iš paveikslo matyti, kad du vartotojai yra prisijungę prie sistemos.

27 pav. Prisijungusių vartotojų paieškos puslapis

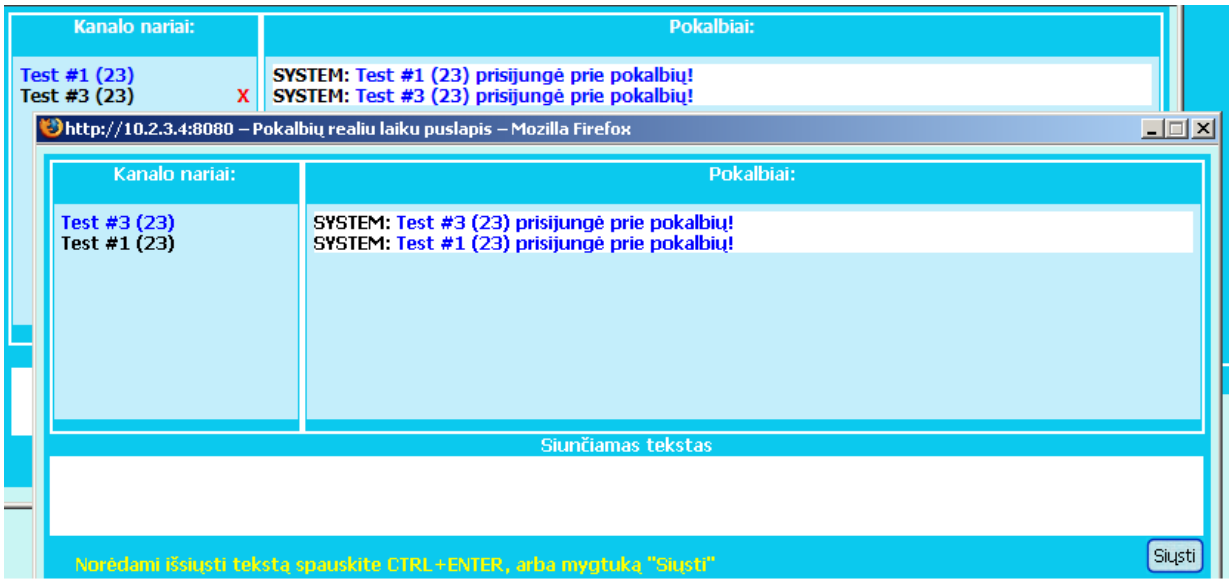
Sekantis, vienas iš svarbiausių puslapių yra **detalios anketos informacijos** rodymo puslapis, kurio pagalba yra rašomi pirmi laiškai kitiems vartotojams, taip pat siunčiami kvietimai bendrauti realiu laiku. Šis puslapis taip pat suteikia galimybę įvertinti tiek anketą, tiek nuotraukas. Ajax naudojamas visuose paminėtose galimybėse.



28 pav. Anketos detalios informacijos rodymo puslapis (dešinysis su įvertinta anketa ir nuotraukomis)

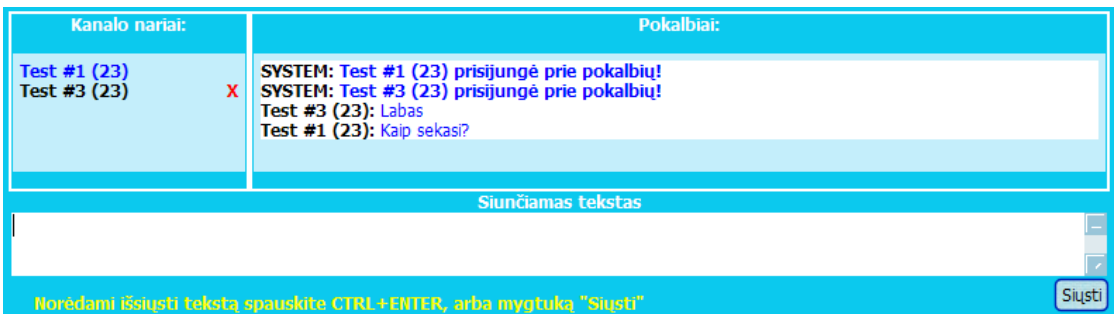
28 paveiksle pavaizduota ta pati anketa, tik dešiniajame pavaizduota anketa po nuotraukų ir pačios anketos įvertinimais. Vertinimas įrašomas pasinaudojus asinchroniniu

kreipiniu į serverį, kuris grąžina suskaičiuotą įvertinimų vidurkį. Šiame puslapyje taip pat galima išsiųsti laišką anketos savininkui (sau vartotojas negali išsiųsti nei laiško, nei vertinimo). Laiškas taip pat siunčiamas asinchroniniu būdu. Paskutinė labai svarbi puslapio funkcija – kvietimo kalbėti realiu laiku galimybė. 28 paveiksle matome pranešimą, kad neatidarytas vartotojo pokalbių puslapis. Norint bendraut reikia tiek siuntėjui, tiek adresatui būti atsidarius **pokalbių** puslapius, kad pokalbiai realiu laiku būtų galimi. Logiška, kad negalima bendrauti tik iš vienos pusės. Turi siunčiamos komandos pasiekti kitą vartotoją, todėl būtina, kad paminėti puslapiai būtų atidaryti. Siuntėjo ir adresato pokalbių puslapiai pavaizduoti 29 paveiksle. Pavaizduotas sėkmingo pokalbio pradžios įvykis.



29 pav. Siuntėjo ir adresato pokalbių puslapiai

Iš 29 paveikslo matyti, kad adresatas priėmė kvietimą ir įvyko sėkmingas kanalo sukūrimas ir vartotojų prijungimas tiek adresato, tiek siuntėjo pusėse. Sistema pranešė, kad abu vartotojai prisijungė prie pokalbių sėkmingai. Po to jau gali abu vartotojai rašyti vienas kitam žinutes (30 paveikslas) . Kviečiantysis vartotojas yra kanalo savininkas, todėl jis gali išmesti iš kanalo nepageidaujamą vartotoją. Tai matyti iš ištrynimo ikonėlės prie „Test #3 (23) “ vartotojo. Taigi ten pavaizduotas kanalo savininko langas, o kvietėjas yra „Test #1 (23) “ vartotojas. Kiekvieno lango savininkas (ne kanalo savininkas) pažymimas pastorintu šriftu.

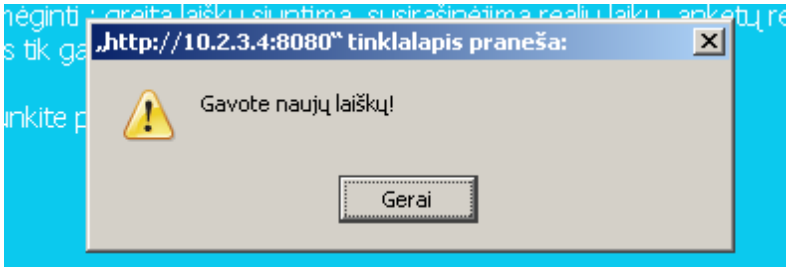


30 pav. Pokalbio realiu laiku dalis

Taigi aptarėme dinamiškus pokalbius. Reikia nepamišti, kad sistemoje galimi ir paprasti laiškų siuntimai, kuriems nereikia, kad adresatas iš vis būtų prisijungęs prie puslapio. Vienas iš rašymo būdų jau buvo aptartas (pavaizduota 28 paveiksle). Liko neaprašytas antras būdas, kai gautas laiškas tiesiog atrašomas siuntėjui. Tai galima padaryti iš **gautų laiškų** puslapio.

30 pav. Gautų laiškų puslapis

Kai vartotojas nusiunčia atsakymą laiško siuntėjui (jei būna prisijungęs) – šis gauna pranešimą (po tam tikro laiko), kad gavo naujų laiškų (31 paveikslas).



31 pav. Gauto laiško (-ų) pranešimas

Taigi aptarėme pagrindinius sistemos puslapius ir jų veikimą. Toliau reikia aptarti konkrečias puslapių savybes ir mėginti jas įvertinti.

4.2 Dinamiškų puslapių savybių tyrimas ir įvertinimas

Pirmiausiai turime nustatyti dinamiško puslapio savybes, pagal kurias jis bus vertinamas:

1) Veikimo sparta.
2) Veikimo aiškumas (paprastumas naudoti).
3) Dizainas.
4) Nedinamiškos savybės.

Prie nedinamiškų savybių galėtume priskirti visas puslapio savybes (funkcijas), kurios nenaudoja Ajax. Bandysime nustatyti kiek puslapyje lieka nedinamiškų savybių, kurios taip pat rodys puslapio dinamiškumo lygį. Vertinimo skalė pasirinkta nuo 0 iki 10. Aptarsime skalės reikšmes kiekvienai savybei.

Pradėsime nuo **veikimo spartos** įvertinimo. Kuo didesnis veikimo spartos įvertinimas – tuo dinamiškesnis yra puslapis. Veikimo sparta ne visada būtinai parodys tik dinamiškų komponenčių veikimą, bet kartu ir nedinamiškų, jei jos yra puslapyje. Tarkime įvertiname veikimo spartą reikšme 0. Tai reikš, kad puslapis veikia labai lėtai, jame visai nėra naudojamas Ajax ir apskritai jis nėra tinkamas naudoti. Jei įvertinimas 10 – puslapis veikia labai greitai, beveik visos jo pagrindinės funkcijos naudoja Ajax. Kiti įvertinimai tarp 0 ir 10 parodys bendrą puslapio veikimo spartos įvertinimą, kartu įvertinus visų funkcijų veikimo spartą. **Veikimo aiškumas** parodo, kiek veikimas yra aiškus vartotojui. Ar sukelia kokių nesuprantamų puslapio veiksmų ir pan. Jei įvertintume puslapio veikimo aiškumą nuliu – puslapio veikimo visiškai nesuprantamas vartotojui. Tai gali sukelti per sudėtingas funkcijų įvertinimas ir pan. Jei įvertiname puslapio veikimo aiškumą 10 – tai reiškia, kad puslapio veikimas yra aiškus ir Ajax naudojimas jame nepasunkino vartotojo supratimo, kaip naudotis sistema ir kokius ji rezultatus pateikia. **Dizainas** parodo, kiek dinamiški puslapio elementai padeda išlaikyti gražų puslapio apipavidalinimą. Kai kada puslapio išvaizda suprastėja dėl dinamiškų jos elementų, bet dažniausiai – pagerėja. Taip pat įvertinimas puslapio dizaino pakeitimo lengvumas, dėl Ajax naudojimo. Tarkime įvertiname dizainą 0. Toks įvertinimas rodo, kad puslapyje sunku ką nors pakeisti ir pats puslapis gali būti realizuotas vien Ajax ir JavaScript pagalba. Priedo pati puslapio išvaizda yra paini ir pan. Jei įvertintume dizainą 10 – tai reiškia, kad Ajax naudojimas puslapio dizainui visai nepakenkė ir jis yra lengvai keičiamas. Kuo didesnis **nedinamiškų savybių** įvertinimas – tuo mažiau dinamiškas yra puslapis, nes jame daugiau yra nedinamiškų elementų, tos pačios informacijos perkrovimas ir apskirtai dažnesnis pačio puslapio persikrovimas. Tarkime įvertiname nedinamiškas savybes 0 – vadinasi visos puslapio funkcijos naudoja Ajax. O tai dar papildomai paspartina puslapio veikimą. Jei įvertintume 10 – tai reikštų, kad puslapio visos funkcijos yra realizuotos nenaudojant Ajax. Tai reiškia, kad, bet koks veiksmas gali iššauks puslapio persikrovimą ir sumažins jo dinamiškumą. Tarpinis įvertinimas parodys funkcijų, nenaudojančių Ajax, santykinį įvertinimą.

Visi eksperimentai buvo atlikti, Windows 2003 Server Standart, Intel Core 2 Duo 6400 @ 2,13 Ghz ir 2 Gb operatyvios atminties, kompiuteryje. Duomenų bazė veikia tame pačiame kompiuteryje. Visi skaičiavimai paimti iš sistemos teikiamos tekstinės veikimo informacijos (logs). Taip pat Tapestry nustatymuose buvo išjungtas puslapių spartinimas, kad būtų galima užfiksuoti didesnius laikus.

8 lentelė. Dinamiškų puslapių savybių įvertinimai

1) Anketos nuotraukų įkėlimo puslapis (26 pav.)
Veikimo sparta – 9
Puslapis veikia labai greitai. Nuotraukos įkėlimo laikas priklauso nuo interneto greičio ir nuotraukos dydžio. Paprasto paveikslo, kurio dydis 120x120 įkėlimas užtruko apie 63 ms. Tos pačios nuotraukos ištrynimo laikas nykstamai mažas. Pats puslapio pirmas užsikrovimas užtrunka apie 235 ms. Pakartotinas apie 200ms. Taigi jei nenaudotume Ajax, nuotraukos ištrynimas užtruktų apie 200 ms. Su Ajax – akimirksniu.
Veikimo aiškumas – 9
Puslapio veikimo principas yra aiškus, bet dėl naršyklės nuotraukų įdėjimo į sparčiąją atminti – norint pamatyti naujai įkeltas nuotraukas reikia pačiam vartotojui perkrauti puslapį. Sistema paryškina perkrovimo užklausimą, kai įkeliama nauja nuotrauka arba sena ištrinama. Jei vartotojas bus nepastabus – nepastebės, kad reikia perkauti puslapį.
Dizainas – 10
Dizainas neperkrautas nereikalingais komponentais. Puslapio išvaizdai Ajax naudojimas nepadarė didelės įtakos. Dizaino atnaujinimai nėra sudėtingi.
Nedinamiškos savybės - 4
Vienintelis nedinamiškas veiksmas – nuotraukų įkėlimas, bet jis realizuotas taip, kad nuotraukų įkėlimo puslapiui nereikėtų persikrauti.
2) Anketos detalios informacijos rodymo puslapis (28 pav.)
Veikimo sparta – 10
Dinamiškumo dėka, puslapis funkcionuoja labai greitai. Visos operacijos dinamiškos. Pats puslapis užsikrauna per apie 140 ms (jei ne savo anketa rodoma) arba apie 125ms (jei savo anketa). Pakartotinis ne savo anketos užkrovimas užtrunka apie 94 ms. Vertinimas užtrunka apie 63 ms. Jei nebūtų puslapis dinamiškas – vertinimas užtruktų apie 130 ms. Laiško siuntimas užtrunka apie 32 ms. Pakartotinas – tik apie 16ms. Jei siuntimas būtų ne asinchroninis – užtruktų apie 115 ms.
Veikimo aiškumas – 9
Vienintelė problema gali iškilti dėl anketos arba jos nuotraukų vertinimo. Vartotojas gali nesusigaudyti, kad reikia paspausti ant žvaigždučių, norint atlikti vertinimą. Ir pagal atitinkama paspaudimą – atliekamas atitinkamas įvertinimas.
Dizainas – 9
Dizainui Ajax įtaka yra labai maža. Dizainą keisti nėra sudėtinga, bet kai kurie komponentai yra sudėtingesni, dėl Ajax naudojimo.

Nedinamiškos savybės - 0

Puslapyje nėra jokių vartotojo veiksmų, kurie priverstu puslapį persikrauti. Viskas veikia dinamiškai (laiškų siuntimas taip pat).

3) Pokalbių puslapis (29 pav.)

Veikimo sparta – 7

Puslapis atsidaro per apie 94 ms. Pakartotinis atidarymas užtrunka apie 78 ms. Atsidarius puslapį, įsijungia serverio apklausinėjimo funkcija, kuri pagal vartotojo aktyvumą keičia apklausinėjimo dažnį, kad subalansuoti naudojamus resursus ir protingą atsako laiką. Serverio apklausimo mažiausias intervalas yra 5 sekundės. Jei būtų didesnis – vartotojas turėtų palaukti per daug ilgai, kad gautų atsaką į jo atliktą komanda, po tam tikro neaktyvumo laiko. Kadangi nėra maksimaliai optimalaus algoritmo apklausinėjant serverį, todėl veikimo spartai skiriamas beveik aukščiausias įvertinimas. Išsiųsta vartotojo žinutė kitą (-us) kanalo vartotojus pasieks priklausomai nuo jų naršyklių serverio apklausinėjimo dažnio, kuris buvo žinutės išsiuntimo metu. Eksperimento metu buvo rašomos žinutės kitiems kanalo dalyviams. Kanale buvo 2 vartotojai. Iš pradžių buvo lėtai rašomos ir siunčiamos žinutės, po to žinučių siuntimo greitis buvo didinamas, po to vėl lėtinamas. Pirmame priede pateiktas grafikas (32 pav.), kuris apytiksliai vaizduoja žinutės atėjimą pas adresatą. Iš pradžių kaip ir minėta, žinutė pasiekia ne iš karto. Didinant žinučių siuntimo grafiką, mažėja laikas, per kurį pasiekia žinutė adresatą. Laikas mažėja, nes kai gaunama žinutė, iškarto tikrinama ar nėra naujų žinučių. Kadangi suveikia žmogiškasis faktorius – laikai nėra labai tikslūs. Žmogus turi parašyti žinutę ir ją išsiųsti. Kol jis parašys naują, kad ir vieno simbolio – laiko praeis pakankamai daug, kad būtų sumažintas apklausinėjimo dažnis. Pats serveris žinutę apdoroja apie 62 ms (pirmo priedo 33 pav.). Tai rodo, kad ilgiausiai užtrunka žinutę transportuoti adresatui. Taigi veikimo sparta yra tik patenkinama. Kompiuterio procesorius net ir vykstant intensyviam žinučių rašymui – nėra apkrautas. Apkrovimo procentas išlieka toks pat, kaip ir nerašant žinutes. Patikrinti sistemos apkrovimo galimybes yra per daug sudėtinga, nes reikia prijungti daugybę vartotojų, o kadangi sistema nėra įdiegta masiniam naudojimui – to padaryti neišeina. Teorinis apkrovimas neparodytų tikros situacijos, nes neįvertintume tinklo pralaidumo ir vėlinimo. Kadangi vykstant pokalbiams, duomenų bazė yra nenaudojama, apkrovimas ateina tik iš žiniatinklio serverio taikomosios programos. Antrame priede pateiktas serverio vidinių žinučių nuorašas, vykstant pokalbiui realiu laiku.

Veikimo aiškumas – 10

Veiksmai elementarūs. Žinutės siuntimas irgi niekuo nesiskiriantis nuo kitų sistemų (programų) žinučių siuntimo principo. Kanalo vartotojo sąrašas taip pat lengvai suprantamas.

	Dizainas – 6
Kadangi dinamiški puslapiai (jų vartotojo sąsaja) keičiami pasinaudojus JavaScript ir DOM technologijomis – puslapio dizainas kuriamas būtent paminėtųjų technologijų pagalba. Tai sudėtingas darbas, nes kiekvienas kintantis puslapio elemento atributas turi būti numatytas. Todėl pakeitimai dizaine yra sudėtingi.	
	Nedinamiškos savybės - 0
Puslapyje nėra jokių vartotojo veiksmų, kurie priverstu puslapį persikrauti. Viskas veikia dinamiškai.	
4) Gautų laiškų puslapis (30 pav.)	
	Veikimo sparta – 7
Puslapis atsidaro per apie 375 ms. Pakartotinis atidarymas trunka apie 200 ms. Laiško siuntimas trunka apie 30 ms. Jeigu siųstume ne asinchroniniu būdų – siuntimo laikas prailgėtų apie 200-300ms. Gautų laiškų puslapio perjungimas trunka apie 200 ms. Laiško šalinimas užtrunka apie 180 ms.	
	Veikimo aiškumas – 10
Standartinis gautų laiškų puslapis.	
	Dizainas – 10
Ajax naudojimas dizainui įtakos nepadarė.	
	Nedinamiškos savybės – 7
Laiškų šalinimas ir laiškų puslapio perjungimas nėra dinamiškas. Jo procese nenaudojamas Ajax. Tai gan stipriai mažina puslapio dinamiškumą.	

4.3. Įvertinimų suvestinė ir išvados

Taigi įvertinome sukurtus dinamiškus puslapius. Galime reziumuoti visus rezultatus.Vertinimų suma sudaroma sudėtus pirmų trijų savybių įvertinimus sumažinus paskutinio vertinimo dydžiu. Vidutinis veikimo greitis nurodo kiek apytiksliai galėtų trukti viena vartotojo operacija (arba jei kelios dinamiškos – imamas jų atlikimo vidurkis), kuri atliekama asinchroniškai. Palyginimui pateikiama apytikrė tos pačios operacijos trukmė, imituojant, kad operacija atliekama be Ajax pagalbos.

9 lentelė. Rezultatų suvestinė

Pus-lapis	Vertinimų suma	Vidutinis puslapio atsidarymo laikas	Vidutinis veikimo greitis su Ajax (ms)	Vidutinis veikimo greitis be Ajax (ms)
1)	**28**	**218 ms**	**63**	**63+218 = 281**
2)	**28**	**133 ms**	**(63+63+32) / 3 = 52**	**52 + 133 = 185**

3)	**23**	**86 ms**	**Nuo 100 iki 2593**	**Nesuskaičiuojama**
4)	**20**	**290 ms**	**30**	**30+290 = 320**

Taigi iš 9 lentelės matome, kad operacijos su Ajax atliekamos daug greičiau. Kiek greičiau, lemia per kiek laiko pasileidžia pats puslapis, nes be Ajax pagalbos, norint atnaujinti dalį puslapio, reikia perkraiti visą puslapį iš naujo. Jei puslapyje daug operacijų – Ajax privalumas nenugingčytinas. Iš vertinimų sumų matome, kad dinamiškiausi yra pirmi trys nagrinėti puslapiai. Vertinimas tik apytikslis, gali nedinamiškas puslapis turėti tokį patį vertinimą, kaip ir dinamiškas, jei nedinamiškame puslapyje bus nedaug funkcijų, arba tos funkcijos bus paprastos ir puslapio persikrovimas bus greitas, arba jo visai nereikės. 4 puslapyje daug nedinamiškų operacijų, kaip puslapio perjungimas, laiško trynimas. Jos labai lėtina šio puslapio funkcionavimą, todėl ir vertinimas yra mažiausias. Pokalbiai realiu laiku be Ajax yra logiškai nesuprantami. Kalbama tik apie Java kalbos internetines sistemas. Galime daryti išvadą, kad sukurta sistema yra dinamiška, nes pagrindinės, dažniausiai naudojamos funkcijos yra sukurtos, naudojant Ajax technologijas. Likusios funkcijos, kurios veikia be Ajax pagalbos neįneša per daug nedinamiškumo sistemai, kad būtų ją galima laikyti nedinamiška.

5. DARBO APIBENDRINIMAS

Keičiasi technologijos, jos tobulėja. Didėja ir žmonių, internetinių sistemų vartotojų, poreikiai. Keičiasi tradicijos. Atsirado Web 2.0 sąvoka. Asmenines svetaines keičia taip vadinamieji blogai. Sparčiai didėja duomenų kiekis. Didėja poreikis juos apdoroti. Seni metodai nebetinka. Taikomosios programos persikelia į internetą. Vis plačiau ir dažniau minimas Ajax technologijų rinkinys (technologija). Ajax tampa neatskiriamas nuo Web 2.0. Dėl didelių, greitai besikeičiančių duomenų kiekių ir vartotojų poreikių augimo – dinamiškos interneto svetainės arba puslapiai tampa vis dažniau reikalingi arba net būtini. Buvo sukurtas ir realizuotas bazinis dinamiškos sistemos (puslapių) modelis. Šis modelis pademonstruoja principinį Ajax veikimą arba kitais žodžiais tariant – kliento naršyklės asinchroninį bendravimą su serveriu. Dauguma karkasų apvelka bazinį Ajax veikimą pagalbinėmis priemonėmis. Šio magistro darbo vienas iš tikslų buvo parodyti bazinius veikimo principus. Modelis realizuotas Java Tapestry ir Hibernate karkasų pagalba. Pagalbiniai karkasai reikalingi tam, kad modelis įgauti normalios sistemos apipavidalinimą. Realioje sistemoje daug geriau parodomi Ajax privalumai ir apskirtai modelis pats savaime tampa daug aiškesnis. Sukurta pažinčių tipo internetinė sistema. Šio tipo sistemos yra labai populiarios tarp interneto lankytojų. Žiūrint į ateitį – sistemą galima išplėsti nuo demonstracinės iki masiškai naudojamos. Sukurta sistema

buvo ištirta pagal 4 savybes : veikimo sparta, veikimo aiškumas, dizainas ir nedinamiškos savybės. Gauti tokie rezultatai kokių ir tikėtasi. Sukurta sistema yra dinamiška. Funkcijos, įgyvendintos pasinaudojus Ajax – veikia daug greičiau. Logiška, nes norint atlikti kokią tai funkciją, susijusią su duomenų apsikeitimu tarp serverio ir kliento naršyklės arba atnaujinti dalį puslapio – be Ajax reikia atnaujinti visą puslapį, kas trunka labai ilgai, lyginant su asinchroniniu bendravimu tarp serverio ir naršyklės. Modelio realizacija pavyko sėkmingai, bet taip pat buvo rastas ir modelio trūkumų. Buvo pasirinktas vienas iš 2 suplanuotų serverio-kliento bendravimo būdų. Buvo naudojamas kintamas serverio apklausinėjimo intervalas. Bet kaip parodė eksperimento rezultatai – tai buvo nelabai geras sprendimas. Net ir esant pakankamai dideliam vartotojo aktyvumui – užlaikymo laikai iki sistemos reakcijos kartais tampa per dideli. Didinti apklausinėjimo dažnio, esant vartotojo neveiksnumui netikslinga, nes tada per daug apkrausime tinklą ir serverį. Kitas modelio trūkumas – vienos žinutės apdorojimas, vieno serverio apklausimo metu. Jei vartotojas staigiai atlieka keletą veiksmų, atsiranda papildomas nedidelis užlaikymas, kol jo atlikti veiksmai atlieka kokius tai pakeitimus. Nes ilgiausiai trunka žinutes (komandos keliavimas) iš serverio pas klientą, nes klientas apklausinėja serverį kintančiu intervalu. Bet reziumiuojant visą darbą – rezultatai tenkina. Buvo sukurta dinamiška sistema, kuri funkcionuoja sukurto dinamiškos sistemos modelio pagrindu, bet masiniam naudojimui reikalingas modelio tobulinimas. Kadangi nebuvo išnaudotos visos galimybės – tai padaryti yra įmanoma ir netgi būtina. Realizuotą sistemą galima naudoti kaip mokomąją priemonę, įvadą į dinamiškų sistemų kūrimą. Reikia dar kartą pabrėžti, kad tai tik vienas iš galimų dinamiškos sistemos modelių, kurį reikia tobulinti ir pritaikyti konkrečiai kuriamai sistemai. Norint išplėsti realaus laiko pokalbių galimybes tiesiog reikia išplėsti komandų sąrašą, kurias priima serveris ir kliento naršyklė. Galima sakyti, kad galimybėms ribų nėra, dar kartą įsitikinę, kad ateityje internetinės sistemos bus beveik visos dinamiškos.

6. IŠVADOS

1) Nustatyta, kad antros kartos interneto atsiradimas atnešė naujas tradicijas, technologijas, didėjančius vartotojų poreikius ir didėjantį dinamiškumo poreikį, kuriam patenkinti dažniausiai naudojamos Ajax technologijos ir papildomi karkasai.
2) Ajax technologijų analizės rezultate buvo pasirinktas toks serverio apklausinėjimo būdas, kai užklausimai siunčiami kintančiu dažniu, priešingai kitam būdui, kai užklausa serveryje laukia, kol ateis nauja komanda. Pasirinktas apklausinėjimo būdas leidžia daug paprasčiau nustatyti, ar klientas dar turi ryšį su serveriu.

3) Pasitelkus Spiar architektūrą, sukurtas bazinis dinamiškos sistemos (puslapio) architektūros modelis, kuris ne tik didina sistemų dinamiškumą, bet ir padeda suprasti bazinius Ajax veikimo principus, kurie yra būtini, norint patobulinti ar išplėsti sukurtą modelį.
4) Sukurta programinė modelio realizacija naudojant papildomus Tapestry ir Hibernate karkasus parodė, kad interneto puslapiai, naudojantys sukurtą modelį, tampa labiau ar net pilnai dinamiški, juose nelieka nedinamiškų savybių.
5) Eksperimentai parodė, kad vartotojas, naudodamasis tokios dinamiškos sistemos funkcijomis, gali sutaupyti vidutiniškai tiek laiko, kiek trunka puslapio atidarymas, kuriame jis naudojasi pasirinkta funkcija, realizuota Ajax pagalba.
6) Darbo naujumas yra tame, kad modelio apdorojamų komandų sąrašas yra lengvai išplečiamas, kas rodo, kad modelio funkcionalumas gali būti lengvai pritaikomas bet kokiai sistemai. Taip pat naujumas yra tame, kad yra naudojami du Ajax varikliai (vienas žinučių siuntimui į serverį, kitas gavimui), o tai leidžia vienu metu siųsti komandas serveriui ir atsinaujinti puslapio informaciją.
7) Po eksperimentų paaiškėjo, kad modelis duoda geras charakteristikas, tačiau jas reikėtų dar patobulinti norint pereiti prie masinio modelio naudojimo realiose sistemose.
8) Šio darbo pagrindu buvo paruoštas ir skaitytas straipsnis „INFORMACINĖ VISUOMENĖ IR UNIVERSITETINĖS STUDIJOS (IVUS‘07)“ konferencijoje.

Literatūra

[1] T. O`Reilly What Is Web 2.0. // Design Patterns and Business Models for the Next Generation of Software. 2005-09-30. Žiūrėta 2007-03-19. Prieiga per internetą http://www.oreillynet.com/pub/a/oreilly/tim/news/2005/09/30/what-is-web-20.html?page=1

[2] Wikipedia. Web 2.0. Žiūrėta 2007-03-19. Prieiga per internetą http://en.wikipedia.org/wiki/Web_2.0

[3] D. Saffer. The Web 2.0 Experience Continuum. // Adaptive Path. 2005 Lapkričio 14. Žiūrėta 2007-03-15. Prieiga per internetą http://www.adaptivepath.com/publications/essays/archives/000545.php

[4] A. Verickas. Web 2.0 – naujoji interneto karta. // Kompiuterija. PC World. 2007 Sausio 15. Žiūrėta 2007-03-19. Prieiga per internetą http://www.kompiuterija.lt/zurnale/2007-01-15/web-20-%E2%80%93-naujoji-interneto-karta/

[5] L. D. Paulson. Building Rich Web Applications with Ajax.. // IEEE JNL. 2005 Spalis. Žiūrėta 2007-03-12. Prieiga per internetą http://ieeexplore.ieee.org/iel5/2/32474/01516047.pdf?tp=&arnumber=1516047&isnumber=32474

[6] A. Mesbah ir A. Van Deursen. An Architectural Style for Ajax.. // IEEE Conference on Software Architecture (WICSA'07). 2007 Žiūrėta 2007-03-12. Prieiga per internetą http://ieeexplore.ieee.org/iel5/11118/35633/01690173.pdf?tp=&arnumber=1690173&isnumber=35633

[7] N. Hanakawa, N. Ikemiya. A web browser for Ajax approach with asynchronous communication model. //IEEE CNF. 2006 Gruodis. Žiūrėta 2007-03-12. Prieiga per internetą http://ieeexplore.ieee.org/iel5/4061321/4061322/04061478.pdf?tp=&arnumber=4061478&isnumber=4061322

[8] K. Smith. Simplifying Ajax-Style Web Development. // IEE JNL, Web Tecthnologies. 2006 Gegužė. Žiūrėta 2007-02-14. Prieiga per internetą http://ieeexplore.ieee.org/iel5/2/34216/01631955.pdf?tp=&arnumber=1631955&isnumber=34216

[9] J. Stamey, T. Richardson. Middleware Development with AJAX. //Journal of Computing Sciences in Colleges. 2006 Gruodis. Žiūrėta 2007-03-19. Prieiga per internetą http://portal.acm.org/ft_gateway.cfm?id=1181948&type=pdf&coll=Portal&dl=GUIDE&CFID=14022313&CFTOKEN=82256928

[10] Wikipedia.Ajax (programming). Žiūrėta 2007-03-20. Prieiga per internetą http://en.wikipedia.org/wiki/AJAX

[11] J. J. Garrett. Ajax: A New Approach to Web Applications. // Adaptive Path. 2005 Vasario 18. Žiūrėta 2005-12-03. Prieiga per internetą http://www.adaptivepath.com/publications/essays/archives/000385.php

[12] D. Crane, E. Pascarello ir D. James. Ajax In Action. // Manning Publications Co. 2005. (119-159p).

[13] A. Bosworth. 10 Places You Must Use Ajax. // Alex Bosworth's Weblog. 2005 Gruodžio 1. Žiūrėta 2005-12-04. Prieiga per internetą http://www.sourcelabs.com/blogs/ajb/2005/12/10_places_you_must_use_ajax.html

[14] R. Smith. Jakarta Tapestry. Žiūrėta 2005-11-30, publikuota 2004-12. Prieiga per internetą <*http://www.ociweb.com/javasig/knowledgebase/2004-12/TapestryPresentation.pdf*>

[15] D. Shrestha and S. Girumala. Žiūrėta 2005-12-01. Prieiga per internetą <*shttp://triton.towson.edu/~schmitt/617/tapestry.html*>

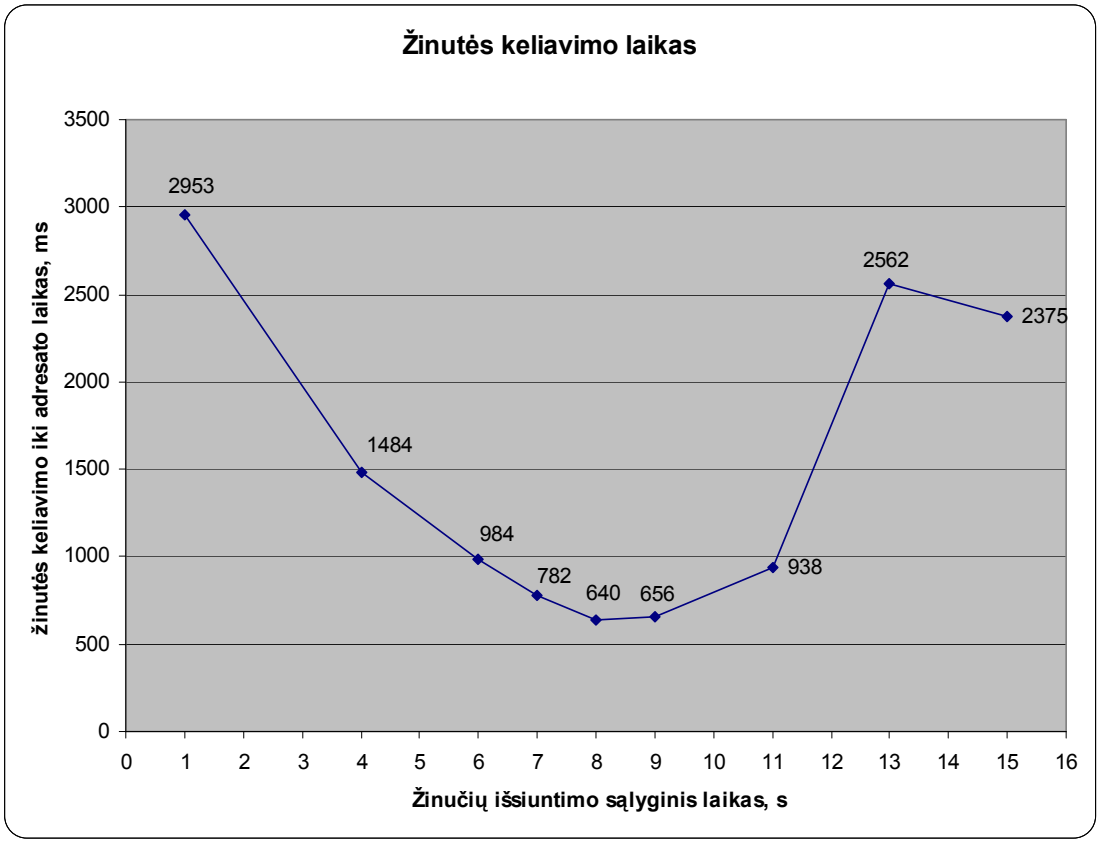
[16] Tapestry namų puslapis. Žiūrėta 2007-03-27. Prieiga per internetą <http://tapestry.apache.org/tapestry4.1/>

[17] Wikipedia. JSON. Žiūrėta 2007-03-27. Prieiga per internetą < http://en.wikipedia.org/wiki/JSON>

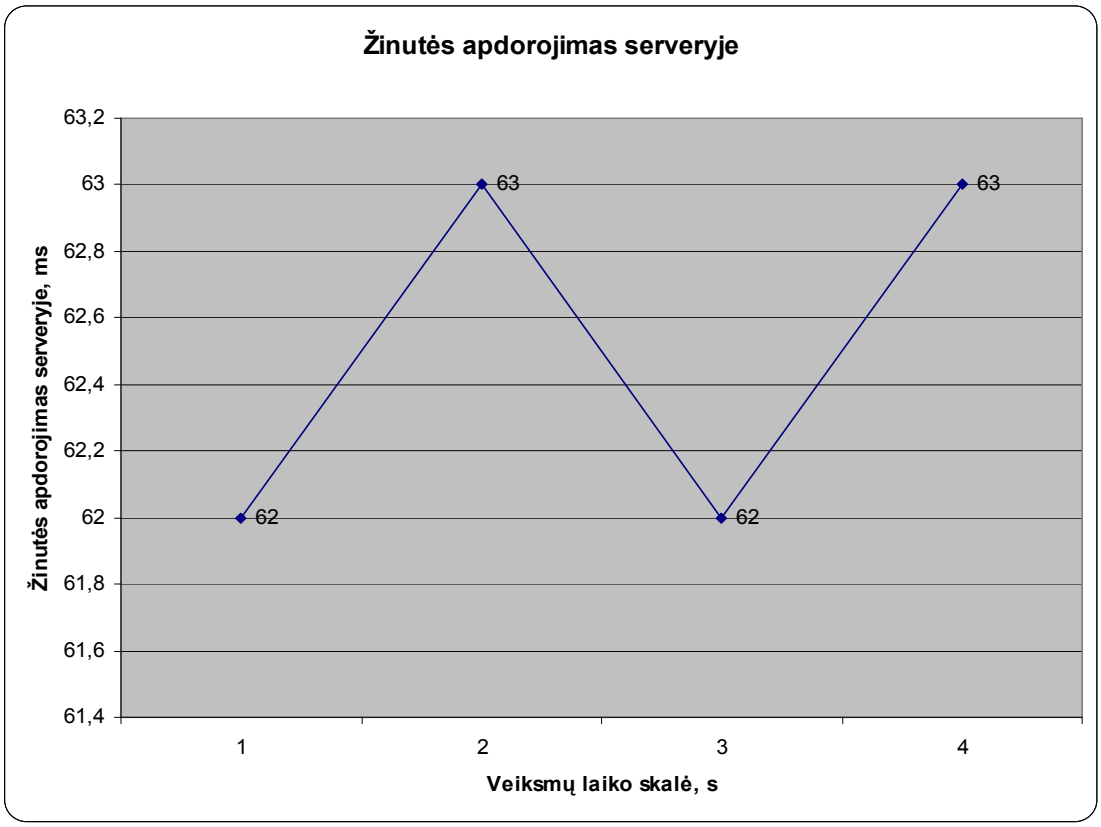
[18] Knyga pdf formatu. H. M. Lewis Ship. Tapestry In action the definitive guide to web application development with Tapestry . Išleista Manning Publications Co .2004 metais

Priedai

Priedas 1. Pokalbių skaičiavimo grafikai



32 pav. Žinutės keliavimo iki adresato laiko grafikas



33 pav. Žinutės apdorojimo serveryje grafikas

Priedas 2. Pokalbių sistemos vidinių veiksmų sekimo (logs) ištrauka

```
INFO  com.dz.tapestry.web.services.Sender [07 16:47:35] =" GET SENDER=  TO 778
1178549255093 "
GOT MESSAGE = *IN*CH*USR*|B749F36AAA62D64A7FFA9EF9873833E2|778 at 1178549255312
SENDING MESSGAE TO USER WITH ID= 778, MESSAGE=
CHAT_INVITATION|B749F36AAA62D64A7FFA9EF9873833E2
INFO  com.dz.tapestry.web.services.Sender [07 16:47:36] =" GET SENDER=  TO 1
1178549256062 "
INFO  com.dz.tapestry.web.services.Sender [07 16:47:37] =" GET SENDER=
CHAT_INVITATION|B749F36AAA62D64A7FFA9EF9873833E2 TO 778 1178549257109 "
INFO  com.dz.tapestry.web.services.Sender [07 16:47:37] =" SENDING COMMAND=
DO_ACCEPT_INVITATION|B749F36AAA62D64A7FFA9EF9873833E2|Test #1 23 (Burundukas)|1 TO
USER= 778 1178549257109 "
GOT MESSAGE = ACCEPT_INVITATION|B749F36AAA62D64A7FFA9EF9873833E2|1 at 1178549259046
INFO  com.dz.tapestry.web.services.Sender [07 16:47:39] =" GET SENDER=  TO 1
1178549259140 "
INFO  com.dz.tapestry.web.services.Sender [07 16:47:39] =" GET SENDER=
ACCEPT_INVITATION|B749F36AAA62D64A7FFA9EF9873833E2|1 TO 778 1178549259140 "
INFO  com.dz.tapestry.web.services.Sender [07 16:47:39] =" GET SENDER=
JOIN_USER|B749F36AAA62D64A7FFA9EF9873833E2 TO 778 1178549259562 "
INFO   com.dz.tapestry.web.services.Sender [07 16:47:39] =" GET SENDER=   TO 778
1178549259609 "
INFO   com.dz.tapestry.web.services.Sender [07 16:47:47] =" GET SENDER=   TO 778
1178549267171 "
GOT MESSAGE = MESSAGE_AWAY|3404C30C24E23960FD537994A6184734|test1 at 1178549267828
INFO  com.dz.tapestry.web.services.Sender [07 16:47:47] =" GET SENDER=
MESSAGE_AWAY|3404C30C24E23960FD537994A6184734|test1 TO 778 1178549267890 "
Message send to client at 1178549267890
INFO  com.dz.tapestry.web.services.Sender [07 16:47:48] =" GET SENDER=
SEND_CHAT_MESSAGE|Test #3 (23)|test1 TO 1 1178549268140 "
Adresata pasiekia zinute 1178549268140
INFO  com.dz.tapestry.web.services.Sender [07 16:47:53] =" GET SENDER=  TO 1
1178549273187 "
INFO  com.dz.tapestry.web.services.Sender [07 16:47:53] =" GET SENDER=
SEND_CHAT_MESSAGE|Test #1 (23)|2 TO 778 1178549273218 "
Adresata pasiekia zinute 1178549273218
GOT MESSAGE = MESSAGE_AWAY|B749F36AAA62D64A7FFA9EF9873833E2|3 at 1178549273421
INFO  com.dz.tapestry.web.services.Sender [07 16:47:53] =" GET SENDER=
MESSAGE_AWAY|B749F36AAA62D64A7FFA9EF9873833E2|3 TO 1 1178549273484 "
Message send to client at 1178549273484
INFO      com.dz.tapestry.web.services.Sender   [07   16:47:53]   ="   GET   SENDER=
SEND_CHAT_MESSAGE|Test #1 (23)|3 TO 778 1178549273656 "
GOT MESSAGE = *CL*CH*WIN* at 1178549277343
Exiting CHAT Window
INFO  com.dz.tapestry.web.services.Sender [07 16:47:57] =" GET SENDER=
CHANNEL_CLOSING TO 778 1178549277734 "
INFO  com.dz.tapestry.web.services.Sender [07 16:47:58] =" GET SENDER=  TO 778
1178549278265 "
INFO  com.dz.tapestry.web.services.Sender [07 16:47:58] =" GET SENDER=  TO 778
1178549278281 "
GOT MESSAGE = *CL*CH*WIN* at 1178549278859
Exiting CHAT Window
```

Priedas 3. Straipsnis, skaitytas „INFORMACINĖ VISUOMENĖ IR UNIVERSITETINĖS STUDIJOS (IVUS‘07) „ konferencijoje.

DINAMIŠKŲ INTERNETO SISTEMŲ KŪRIMO METODIKA

Darius Žilinskas

Kauno Technologijos Universitetas, K. Donelaičio g. 73, LT-44029 Kaunas

Šiame straipsnyje bus kalbama apie antros kartos (Web 2.0) interneto atsiradimą, jam būdingas naujas technologijas ir pasikeitusius reikalavimus interneto sistemoms. Pastebimas augantis poreikis darbui su dideliais duomenų kiekiais, naudojant paprastą naršyklę. Iškeliamas tikslas pasiekti tokią pat veikimo spartą kaip ir darbastalio programų. Detaliau aptariamas technologijų rinkinys, pavadintas Ajax (Asynchronous JavaScript and XML), kuris leidžia kurti dinamiškus interneto puslapius (sistemas). Pateikiamas dinamiškos sistemos apibrėžimas ir bazinis įgyvendinimo modelis (remiantis Spiar architektūra), kuris atskleidžiantį esminius Ajax veikimo principus.

Įvadas

Beveik kiekvienas šiuolaikinis žmogus nebeįsivaizduoja savo gyvenimo be interneto. Internetas tampa nebeatskiriama gyvenimo dalimi. Taip nutiko todėl, kad kaip ir technologijos tobulėja – tobulėja ir interneto teikiamų paslaugų spektras bei kokybė, daugėja vartotojų reikalavimų. Atsiranda tokios sąvokos kaip Web 2.0 (antros kartos internetas), dinamiški puslapiai (sistemos), Ajax ir kitos. Interneto vartotojai nori iš interneto gauti vis daugiau ir daugiau paslaugų. Sparčiai didėja duomenų kiekiai, kuriuos reikia pateikti vartotojui ir leisti juos keisti. Todėl natūralu, kad reikalingos naujos technologijos ir modeliai, tokiems informacijos kiekiams apdoroti. Vartotojas nenori ilgai laukti, jis atmes sistemą kaip netinkamą, jei ši veiks lėtai, todėl šiame straipsnyje bus kalba apie dinamiškas sistemas, jų poreikį. Bus suprojektuotas dinamiškos sistemos bazinis modelis, kuris pademonstruos, kaip gali atrodyti dinamiška sistema ir kaip ji veikia.

1. Dinamiškų sistemų (puslapių) kūrimo poreikis ir technologijos

Šiame skyriuje aprašomas antros kartos internetą, jo atneštus naujus poreikius interneto sistemoms ir pagrindines technologijas, kurios bus reikalingos (kai kurios net būtinos) kuriant dinamiškas sistemas.

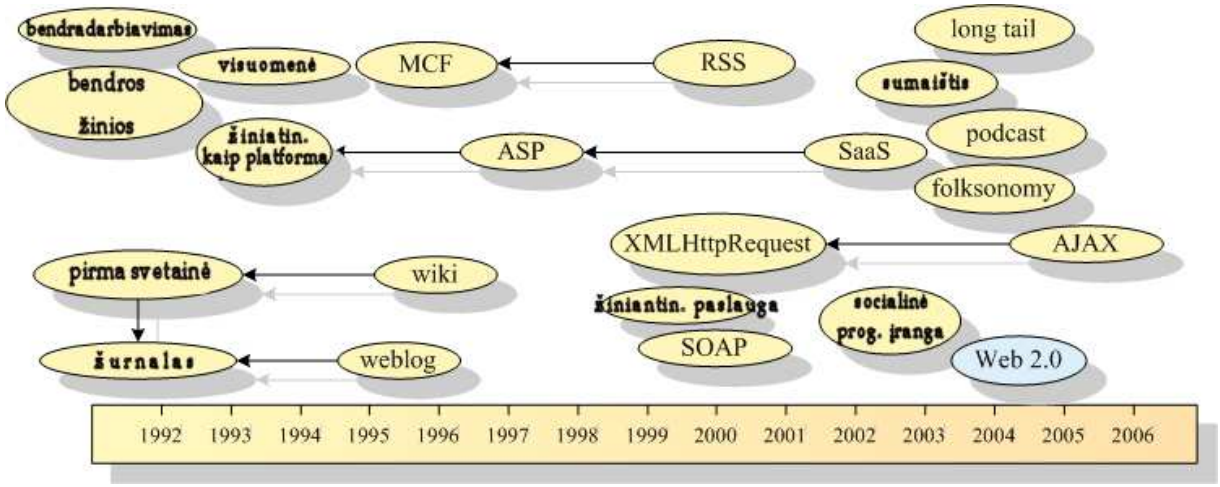
1.1 Dinamiškumo apibrėžimai

Dinamiška sistema (puslapis) – tai sistema (puslapis), kurios (-io) daugumos funkcijų veikimas didžiąja dalimi paremtas asinchroniniu bendravimu su serveriu arba kitaip tariant – naudoja Ajax technologijas.

Dinamiškos sistemos (puslapio) modelis – architektūrinis Ajax naudojimo modelis, kurio pagalba sistemos funkcijas bus galima įgyvendinti Ajax technologijų pagalba.

1.2 Antros kartos internetas Web 2.0

Kaip teigia Tim O`Reilly [6], interneto „burbulas “ sprogo 2001 metais. Šis sprogimas pažymėjo interneto pasikeitimo tašką. Web 2.0 konceptas buvo pradėtas konferencijoje kolektyviniame naujų idėjų svarstyme tarp O`Reilly ir MediaLive International [6]. Pirmame paveiksle pavaizduotas Web 2.0 atsiradimas.



1 pav. Web 2.0 atsiradimas bendroje laiko skalėje [7]

Dale Dougherty, kuris yra interneto pionierius ir O`Reilly VP, pažymėjo, kad internetas tapo dar svarbesniu, nei bet kada anksčiau, reguliariai atnešdamas vis naujas svetaines ir stulbinančias taikomąsias programas [6]. Toliau panagrinėsime konkretesnius Web 2.0 apibrėžimus. Lietuva irgi neatsilieka nuo pasaulio ir nagrinėja, bei pastebi Web 2.0 privalumus ir naujoves. [2] šaltinis patvirtina, kad Web 2.0 termino pradininkai yra paminėtieji O`Reilly Media group. Kalbant apie pačio termino apibrėžimą, susiduriame su problema. Kaip teigia [2] šaltinis, terminas yra įvairiai interpretuojamas ir pati jo reikšmė nėra tiksliai apibrėžta. Vieną iš apibrėžimų pateikia Wikipedia [7]: Web 2.0 – tai antros kartos internetu paremtos paslaugos: socialinės-tinklinės svetainės, wiki, komunikavimo įrankiai bei vartotojų sukurtos sistemos, kurios skirsto į kategorijas interneto puslapius (angl. folksonomies). Folksonomy Wikipedia apibrėžia taip [7]: tai vartotojo sukurta sistematika,

naudojama internetinių puslapių, nuotraukų, nuorodų ir kito žiniatinklio turinio gavimui ir skirstymui į kategorijas. Sukurtos sistemos pabrėžia internetinį bendradarbiavimą ir informacijos dalinimąsi tarp vartotojų [7]. Web 2.0 stiprybė yra duomenys. Žiniatinklis tampa lyg platforma, kurios pagalba gali būti kuriamos naujos paslaugos. Kuo daugiau žmonių naudojasi paslauga – tuo didesnę naudą jie gauna. [2] šaltinio teigimu, tai atsispinti pažinčių svetainėse, kurios jau senokai neatitinka tokio primityvaus apibrėžimo. Šios svetainės dabar pirmiausiai suteikia galimybę išreikšti save. Kaip pavyzdys, pateikiama MySpace. (My Space karta). Kaip jau minėta Web 2.0 stiprybė yra duomenys. Svarbu ne tik patys duomenys, bet ir tai, kad Web 2.0 suteikia galimybę dirbti su dideliais duomenų kiekiais. Kaip pavyzdį [2] šaltinis pateikia Wikipedia, kuri pagal duomenų kiekį yra spausdintos Britannica enciklopedijos versijos varžovė. Asmeninius puslapius keičia asmeniniai dienoraščiai (angl. blog). Tokius dienoraščius yra patogu valdyti, keisti, atnaujinti ir t.t. Taip pat [2] šaltinis teigia, kad dar viena labai svarbi Web 2.0 savybė yra nuolatinis tobulinimas (angl. the perpetual beta). Vartotojai turi nuolat ieškoti naujų paslaugų pritaikymo būdų. Tos paslaugos gali būti net nenumatytos kūrėjų, bet jie nuolat tobulina kodą ir siūlo naujas funkcijas.

Išvardinsime technologijas, kurias Web 2.0 svetainė paprastai tūrėtų naudoti. [7] šaltinis pateikia tokį sąrašą :

- Turtingos (rich) internetinės taikomosios programos, neprivalomai paremtos **Ajax**.
- CSS (Cascading Style Sheets).
- Semantiškai teisingos XHTML žymės ir mikro-formatų naudojimas (mikro-formatų žymės tai specifiniai HTML atributai: class, rel, rev).
- RSS/Atom viduje esančių duomenų surinkimas ir įgalinimas dalinti (-is) ją.
- Aiškūs ir suprantami URL.
- Didelis folksonomies naudojimas (pvz. žymių forma).
- Dalinis arba pilnas wiki programinės įrangos naudojimas (dalinis naudojimas gali išaugti iki pilnos platformos, skirtos svetainei).
- Internete sugeneruotų (angl. webblog) svetainių publikavimas.
- REST arba XML interneto paslaugų APIs.

1.3 AJAX (Asinchroninis JavaScript ir XML)

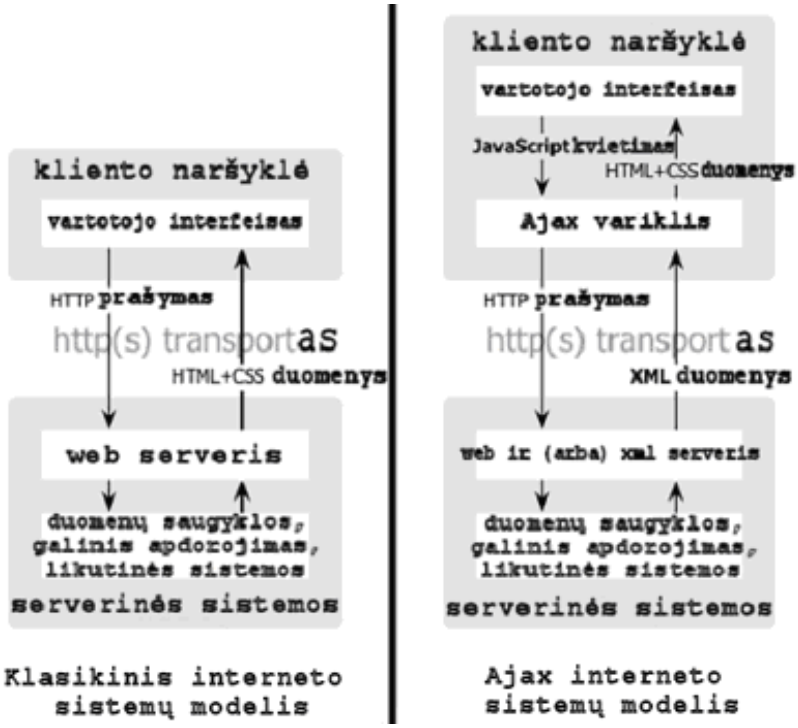
Šiame skyrelyje aprašomas pagrindinis dinamiškos sistemos technologijų rinkinys, pavadintas Ajax. Realiai jokios tikros dinamiškos sistemos negali nenaudoti Ajax, nes tai pagrindinė priemonė pagreitinti serverio ir kliento bendravimą.

1.3.1 Architektūra, veikimo principai

Asynchronous JavaScript and XML, arba Ajax, yra žiniatinklio plėtojimo technika, skirta kurti interaktyvioms interneto taikomosioms programoms [8]. Realiai Jesse James Garrett [3] teigia, kad Ajax nėra tik technologija. Realiai tai yra keletas technologijų, kur kiekviena iš jų veikia savaip, bet kartu sudėjus gali būti panaudota labiau efektyvesniems tikslams.

Ajax jungia tokias technologijas :

- XHTML (or HTML) ir CSS informacijos žymėjimui ir stilizavimui [8].
- Dokumentų Objektų Modelis (DOM) manipuliuojamas panaudojant JavaScript, kad dinamiškai atvaizduoti pristatomą informaciją ir sąveikauti su ja [8].
- XMLHttpRequest objektas, naudojamas asinchroniniam duomenų pasikeitimui su serveriu [8].
- Duomenų mainams ir manipuliacijoms naudojama dar XML ir XSLT [3].
- JavaScript, kuris jungia viską į vieną krūvą [3].



2 pav. Klasikinės ir Ajax internetinių sistemų modeliai [3]

Klasikinis Web taikomųjų programų modelis dirba taip. Dauguma vartotojų veiksmų vartotojo sąsajoje sužadina HTTP užklausą atgal į web serverį. Tada serveris atlieka kažkokius veiksmus ir grąžina rezultatus į į HTML puslapį klientui. Šis technologinis sprendimas gal ir turi techninę prasmę, bet vartotojas turi laukti, kol serveris apdoros užklausimą [3]. Kyla klausimas, kam vartotojui iš viso matyti, kaip taikomoji programa kreipiasi į serverį iš vis. Štai šioje vietoje ir turime Ajax, kuris panaikiną pradėti-sustoti-pradėti-sustoti sąveikos prigimtį internete bei pristato savo Ajax variklį tarp vartotojo ir serverio. Ajax variklis vartotojui leidžia sąveikauti su taikomąja programa asinchroniškai ir nepriklausomai nuo komunikacijos su serveriu. Taigi vartotojas niekada nelaukia, žiūrėdamas į tuščią ekraną, kol serveris kažką daro. Kiekvienas vartotojo veiksmas vietoj to, kad paprastai sugeneruoti HTTP užklausą – perduos JavaScript formos šaukinį (angl. call) į Ajax variklį. Bet koks atsakas į vartotojo veiksmą nereikalauja sugrįžimo į serverį (pvz., paprasta laukų reikšmių patikra) [3]. Ajax svetainės gali siųsti užklausas serveriui ir gauti atsakymą tam tikra apsibrėžta forma (SOAP ar kita XML paremta), bei naudojant JavaScript programavimą atnaujinti tik reikiamą puslapio dalį [8].

Puslapiai, sukurti naudojant AJAX technologija, reikalauja naršyklių, palaikančių šias technologijas. Tokios naršyklės yra Mozilla Firefox, Internet Explorer, Opera, Konqueror ir Safari [8]. Ajax jau yra plačiai pradedamas naudoti. Google deda dideles investicijas į Ajax vystymą. Orkut, Gmail, naujausia Google Groups versija, Google Suggest ir Google maps yra Ajax taikomosios programos. Flickr priklauso nuo Ajax, Amazon A9.com paieškos variklis naudoja panašias technologijas [3].

1.3.2 Ajax privalumai ir trūkumai

AJAX privalumai:

- Ajax leidžia Web taikomosioms programoms atsiliepti daug greičiau į daug skirtingų vartotojo užklausų ir išvengti pakartotino nepasikeitusios informacijos siuntimo tinklu. Kadangi Ajax technologijos yra atviros, jos yra palaikomos visų naršyklių, kuriose veikia JavaScript, nepriklausomai nuo operacinės sistemos tipo [8].
- Nuolatinė komunikacija su serveriu [4].
- AJAX yra suderinamas su daugelio tarpinės programinės įrangos apibrėžimais (angl. defininion) [4].
- Panaikinami pilni puslapių atgaliniai siuntimai (angl. post-back), pritaikius mažesnius, didėjančius atnaujinimus [5].
- Turi įtakos kliento mašinos apdorojimo greičiui (galiai), leidžiant naršyklei būti atsakingu už daugiau taikomosios programos vykdymo aspektų [5]. Taip nuimama apkrova nuo serverio.
- Išnaudojamos šiuolaikinės grafikos galimybės: permatomumas, šešėliavimas, animacija, objektų gylio nustatymai ir t.t. Suteikiama daugiau interaktyvumo informacijos pristatymui [5].

AJAX trūkumai:

- JavaScript kalbos apribojimai [5].
- Naršyklių nesuderinamumai [5].

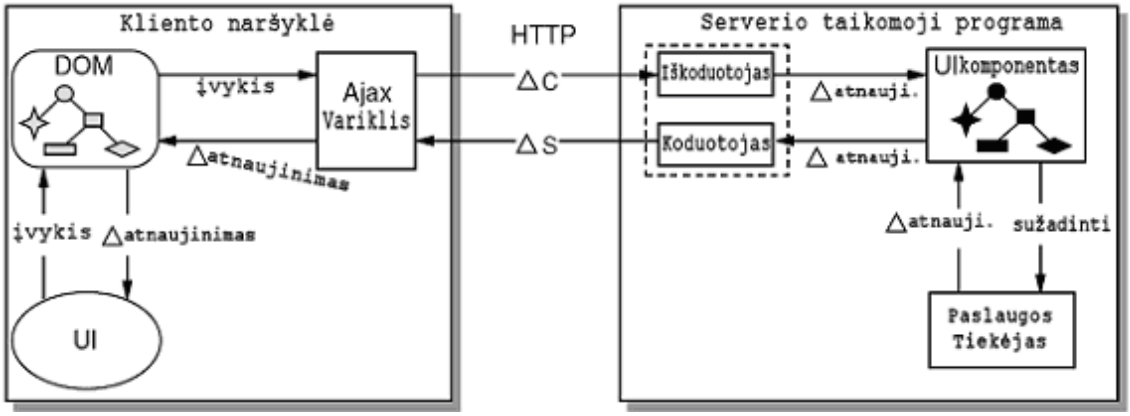
- „Back “ naršyklės mygtuko problema [5].
- Problema, susijusi su puslapio pažymėjimu (angl. bookmark), kadangi puslapis tampa labai dinamiškas [8].
- Tinklo gaišaties laikas arba intervalas tarp vartotojo užklausos ir serverio atsakymo turi būti atidžiai apgalvotas Ajax taikomose programose. Vartotojai gali nesuprasti, kodėl nieko neįvyko ar kodėl kažkas įvyko ne tuo laiku, kad jie to tikėjosi. Ypač jei tai susiję su vartotojo sąsajos pakeitimais [8].
- Kaip ir DHTML svetainėms, AJAX svetainėms reikia žymiai daugiau testavimo, nes kiekviena naršyklė gali elgtis šiek tiek skirtingai [8].
- Naudojant AJAX, padaugėja mikro-užklausų į serverį, todėl neapgalvotai naudojant gali sulėtinti bendravimą tarp kliento ir serverio [8].
- Ajax nepritaikytas žmonėms su negalia, nes yra orientuotas tik į grafinę sąsają turinčias naršykles [8].

2. Dinamiškos sistemos (puslapio) bazinis Ajax modelis

Šiame skyriuje bus aprašomas bazinis dinamiškos sistemos (puslapio) modelis, naudojantis Ajax technologijų rinkinį ir SPIAR arvhitektūros pagrindus. Taip pat pateikiami sukurtos eksperimentinės sistemos, kuri naudoja aprašytąjį modelį, eksperimentų rezultatai.

2.1 Modelio aprašymas

SPIAR tipo architektūra geriausiai parodo Ajax veikimo pagrindus, tuo pačiu aiškiai matomas dinamiško puslapio veikimo principas. Be to, ši architektūra labiausiai tinka dinamiškos internetinės svetainės modelio sukūrimui. SPIAR architektūra daugiau orientuota vienam puslapiui, o ne visai sistemai. Bet panaudojus pakartotinį panaudojimą ir komponentinio kūrimo galimybę (Pvz. Tapestry karkasą), modelį galime išplėsti visai sistemai. 3 paveiksle pavaizduotas SPIAR Ajax architektūros principinė vaizdavimo schema.

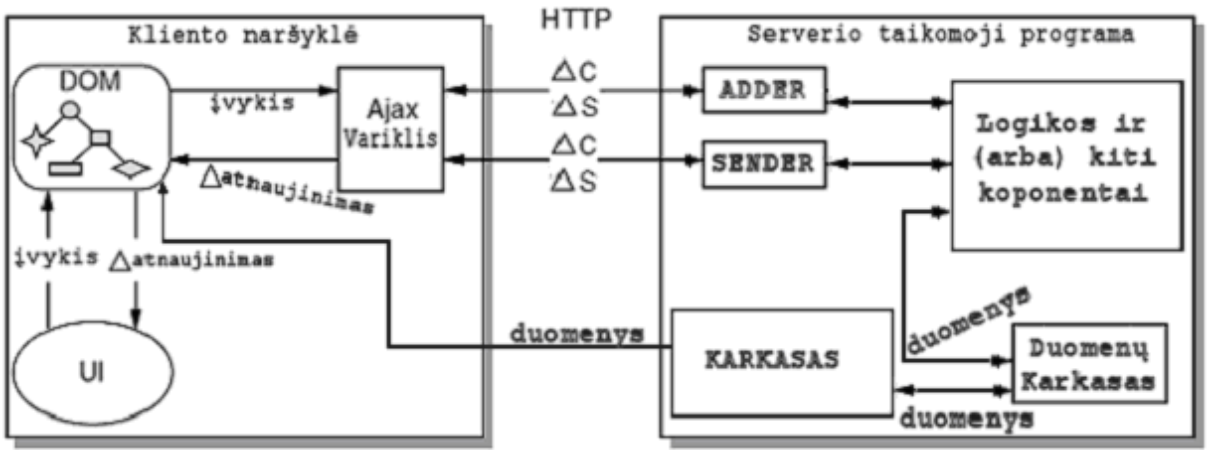


3 pav. SPIAR Ajax architektūra [1]

Pagrindiniai SPIAR architektūros elementai yra sudalinti į tris kategorijas: apdorojimo, duomenų ir sujungimo. **Apdorojimo elementai** – tai tokie komponentai, kurie suteikia duomenų elementų transformacijas. **Duomenų elementai** – tokie elementai, kurie turi savyje informaciją, kuri yra naudojama ir transformuojama apdorojimo elementų. **Sujungimo elementai** – tai elementai, kurie laiko komponentus kartu, suteikiant jiems galimybę komunikuoti tarpusavyje [1].

Pačios architektūros veikimo principas yra nesudėtingas. Vartotojas atlieka veiksmus vartotojo sąsajoje, tie veiksmai sužadina įvykius, kurie Ajax variklio pagalba ir ΔC (kliento teikiama informacija) sujungėjo pagalba nusiunčia užkoduotus duomenis serveriui, kuris iškoduotojo pagalba sužadina reikiamus tų įvykių apdorojimus UI komponentuose. Pvz. duomenų nuskaitymas iš duomenų bazės ir panašiai. Serveris sugeneruos atsakymą ir, užkodavęs su koduotoju, nusiųs klientui pasinaudojęs ΔS (serverio teikiama informacija). Kliente esantis Ajax variklis nusiųs atnaujinimo komandą DOM, kurio pagalba bus atnaujinta vartotojo sąsaja [1].

SPIAR architektūrą reikia pritaikyti mūsų kuriamam dinamiškos sistemos modeliui. Daugiausiai pakeitimų reikės atlikti serverio pusės architektūroje, nes bus stengiamasi suprojektuoti taip, kad kuriama dinamiško puslapio architektūra tarnautų ir kitoms dinamiškos sistemos dalims. Kadangi Ajax variklis gali būti bet kokiame kliento naršyklei pasiekiamame sistemos puslapyje – serverio dalis taip pat gali būti pasiekiama nebūtinai tik iš vieno puslapio, turinčio Ajax variklį. Tai labai patogi galimybė, nes galima sukurti visą logiką tuose pačiuose komponentuose, tereikia iškoduoti/užkoduoti naujas žinutes. Apačioje esantis paveikslas parodo abstrakčią dinamiško puslapio architektūrą, kuri realizacijoje gali būti pritaikyta konkrečiai kuriamai dinamiškai sistemai.



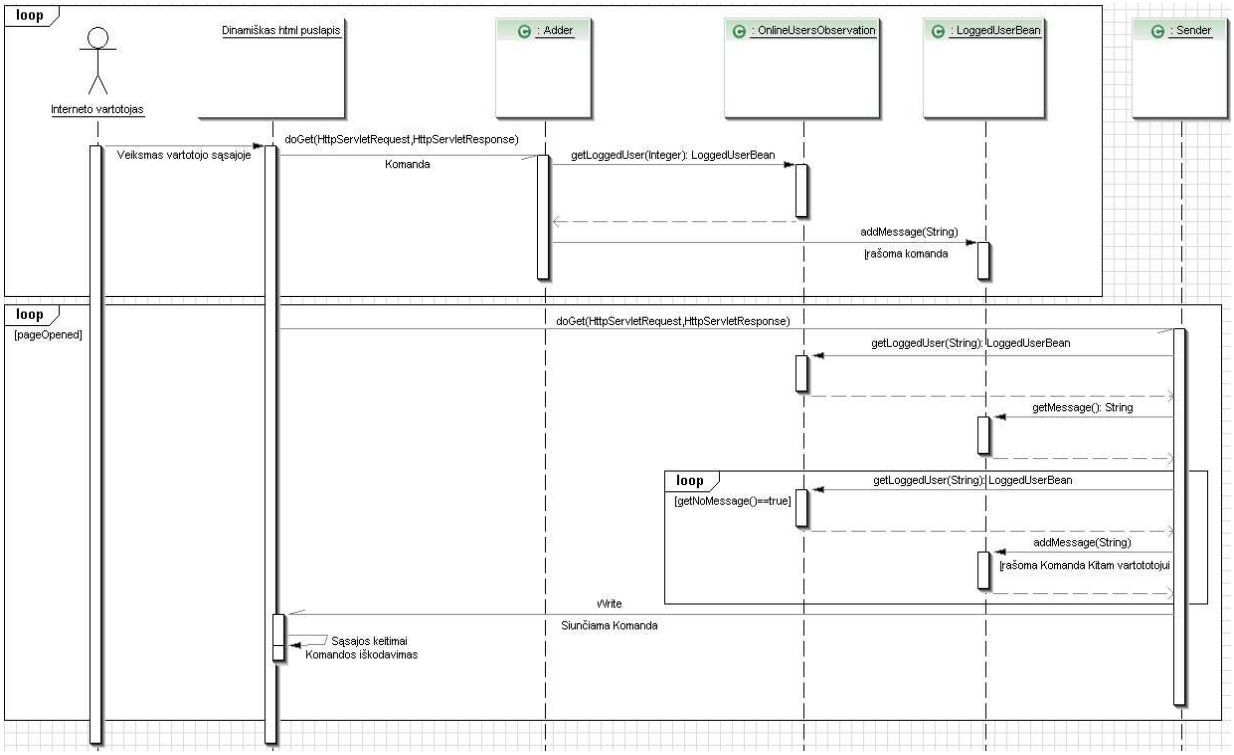
4 pav. Bazinė dinamiško puslapio architektūra

Kadangi dauguma puslapių yra kuriami pasinaudojus pagalbinius karkasus – architektūra papildoma duomenų „ Duomenų Karkasas“ ir internetinių sistemų kūrimo karkasais „ KARKASAS“. Duomenų karkaso pavyzdys gali būti Hibernate karkasas, o sistemos: Tapestry, Struts, Spring ir kiti. Logikos ir kiti komponentai gali būti realizuoti pačiame karkase, bet Ajax veikimo principų parodymui logika yra išskirta. „ADDER“ atitinka „ Iškoduotojas“ ir yra skirtas priimti žinutes iš kliento naršyklės. Papildomai pridedama galimybė žinutes apdoroti pačiam ir siųsti atsakymą atgal. Taip padaroma todėl, kad kai kurie veiksmai reikalauja labai mažai logikos ir atsakymas į juos gali būti taip pat labai paprastas. Bet pagrindinė „ADDER“ paskirtis lieka iškoduoti ateinančias žinutes ir atitinkamai iššaukti veiksmus. „ SENDER“ atitinka „ Koduotojas“. Jo tikslas yra siųsti užkoduotas komandas kliento naršyklei, kad Ajax variklis galėtų atlikti atitinkamus veiksmus vartoto sąsajai. Kliento naršyklė per Ajax kreipiasi į „SENDER“, kad šis jai atsiųstų komandas. Priklausomai nuo logikos, komandų gali prisikaupti ir jas būtina sudėti į eilę. Darome prielaidą, kad vieno kreipinio į „SENDER“ metu galima gaut tik vieną užkoduotą žinutę iš serverio. Kliento naršyklėje esantis puslapis yra sudarytas iš vartotojo sąsajos elementų, kuriuos galima valdyti DOM ir JavaScript pagalba. Dinamiškame puslapyje turi būti suprojektuota logika, kuri Ajax variklio pagalba būtų susieta su severio pusėje esančia logika. Susiejimas susideda iš galimybės siųsti žinutes serveriui ir galimybės apdoroti atėjusius atsakymus. Taip pat turi būti numatytas kreipinių dažnio reguliavimas pagal veiksmų dažnumą.

Galimi du variantai apklausinėjant serverį:

3) Nusiuntus užklausimą į serverį laukti atsakymo, kol jis bus suformuotas ir išsiųstas. Šis požiūris turi problemą. Serveris yra apkraunamas procesais, laukiančiais atsakymų į užklausimą. Taip pat kyla problema, kada iš vis nutraukti laukimą, jei laukimo laikas viršija nustatytas normas. Viskas vykdoma asinchroniškai, todėl vartotojui laukimas nesutrukdys atlikti kitų veiksmų. Tas pats galioja ir 2 variantui. Šis požiūris turi privalumą, kad vartotojas gauna atsaką iš karto, kai jis tik yra suformuojamas.
4) Nusiuntus užklausimą į serverį nelaukti atsakymo. Taip serveryje nebus sukuriama tiek procesų, kiek ateina užklausų. Kliento naršyklė kintančiu dažniu gali apklausinėti serverį, ar nėra naujų komandų. Toks požiūris turi privalumą, kad serveris nebus apkraunamas laukiančiais procesais. Žinoma, yra ir trūkumų. Reikalinga protinga logika, kuri siuntinės užklausimus serveriui. Jei užklausimų siuntimas nebus kontroliuojamas, gali būti perkrautas tinklas užklausomis kaip ir pats serveris.

Tarkime pasirenkame 2 variantą. Sistemos veikimas tada būtų toks, kaip pavaizduota 5 paveiksle (Darome prielaidą, kad visos žinutės pasiekia serverį akimirksnių, be jokio užlaikymo).

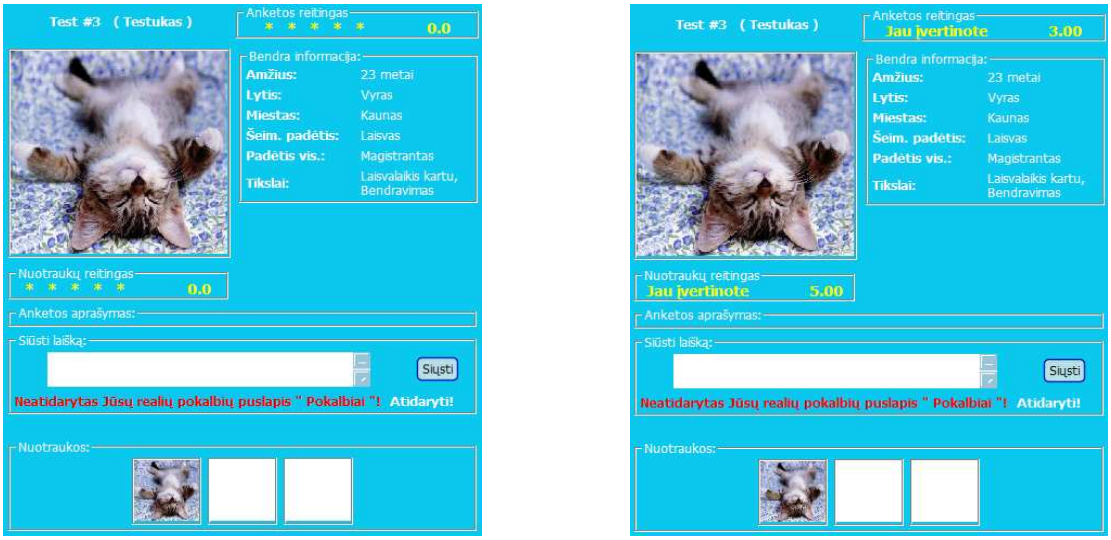


5 pav. Dinamiškos sistemos veikimo sekų diagrama

Kaip galima matyti 20 paveiksle, sistema veikia pilno duplekso principu. T.y. tarp serverio ir kliento sudaromas abipusis ryšys. Naudojami skirtingi Ajax varikliai žinutėms nusiųsti ir gauti. Interneto vartotojas gali atlikti tam tikrus veiksmus sistemos vartotojo sąsajoje, kurie reikalauja sąveikos su serveriu, kai tuo pačiu metu severis gali atsiųsti komandą pakeisti vartotojo sąsajos dalį ir pan. Penktame paveiksle esančioje sekų diagramoje nagrinėjami tik tie veiksmai, kurie reikalauja sąveikos su serveriu. Arba kitais žodžiais tariant – reikalauja asinchroninio kreipimosi į serverį. Kai vartotojas atlieka veiksmą, Ajax variklis nusiunčia užkoduotą komandą į serverį, kur žinutę iškoduoja „Adder " komponentas ir įrašo ją į vartotojo žinučių eilę. Vartotojas gaunamas pagal jo sesijos identifikatorių, pasinaudojus „OnlineUserObservation " komponentu, kuris skirtas grąžinti prisijungusio vartotojo komponentą „LoggedUserBean ". Vartotojo visi atliekami veiksmai bus įrašomi į jo žinučių eilę. Lygiagrečiai puslapis apklausinėja serverį, „Sender " komponentas paima vieną naujausią žinutę iš apklausiančio vartotojo žinučių eilės ir ją apdoroja. Apdorojimo metu gali suformuoti naujas žinutes kitiems vartotojams ir pan. Tai gali kartotis tol, kol nebebus naujų žinučių. Gautą iš serverio žinutę apdoroja puslapio viduje esantis JavaScript kodas. Pagal pasirinktą serverio apklausinėjimo variantą, serveris yra apklausinėjamas kintančiu dažniu. Dažnis mažėja, jei vartotojas neatlieka jokių veiksmų, kurie įneštų naujų žinučių. Dažnis padidėja iki maksimalaus, atsiradus, bet kokiai komandai, skirtai vartotojo sąsajai arba serveriui iš kliento pusės.

2.2 Eksperimentinė modelio realizacija

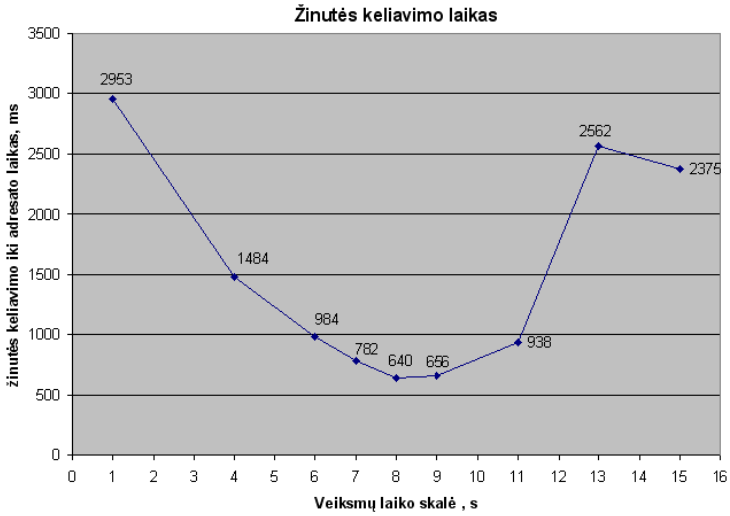
Toks modelis gali būti įgyvendintas realioje sistemoje. Pasirinkę konkrečią platformą, serverio apklausinėjimo būdą (vieną iš pateiktų dviejų arba savo) ir technologijas galima sukurti eksperimentinę arba realią sistemą. Eksperimentinei sistemai buvo pasirinktas Tapestry karkasas. Duomenų valdymui – Hibernate karkasas. Panaudoti tokie logikos komponentai: pokalbių kanalas, prisijungęs vartotojas ir prisijungusių vartotojų sekimo komponentas.



6 pav. Eksperimentinė modelio realizacija pažinčių tipo internetinėje sistemoje

6 paveiksle pavaizduotas vienas iš dinamiškų puslapių, kuris funkcionalumo įgyvendinimui naudoja Ajax. Paveiksle pavaizduotas vartotojo anketos vertinimo procesas pažinčių tipo sistemoje. Kairiajame paveiksle anketa pavaizduota neįvertinta, o dešiniajame – po įvertinimo. Puslapyje visi įmanomi veiksmai realizuoti panaudojus suprojektuotą dinamiškos sistemos modelį. Tai anketos ir nuotraukų vertinimo, laiško siuntimo, kvietimo kalbėti realiu laiku funkcijos. Ajax naudojimas daugiausiai turi įtakos puslapio funkcijų veikimo spartai, pačio puslapio funkcionavimo aiškumui ir dizaino kūrimui. Iš eksperimentų nustatyta, kad puslapio funkcijų veikimo greitis, naudojant Ajax, vidutiniškai apytiksliai padidėja puslapio atsidarymo laiko dydžiu. Veikimo aiškumas realiai priklauso pagrindinai tik nuo klaidų pranešimų, nepavykus Ajax komandai, bet dizainas – priklauso beveik visais aspektais. Serveris asinchroniškai gali atnaujinti kliento puslapį tik Javascript ir DOM pagalba. Tai labai apsunkina puslapio dizaino kūrimą.

Buvo atliekami eksperimentai, kurie nustatytų, kaip laiko atžvilgiu veikia sukurtas modelis. Labiausiai rūpimas klausimas – per kiek laiko vieno vartotojo nusiųsta komanda (eksperimento metu – tekstinė žinutė) pasiekia jos adresatą (pvz. kitą vartotoją). Buvo nustatyti laikai, kurie parodė, kad modelio veikimo laikai yra tokie, kokių ir tikėtasi. Vartotojui aktyviau siunčiant žinutes – jos greičiau pasiekia adresatą (7 pav.). Iš pradžių vartotojas pamažu siuntė žinutes. Po 2 nusiųstų žinučių siuntimo greitis buvo didinamas ir apie 7-9 laiko skalės pozicijose (7 pav.) pasiekė maksimumą. Paaiškėjo, kad net ir kiek įmanoma greitai siunčiant žinutes kitam vartotojui, žinutės jį pasiekia per apie 650 ms, kai realiai pats žinutės apdorojimas serveryje užtrunka apie 60 ms. Kitos operacijos, kurių rezultatai turi būti grąžinti siuntėjui, vykdomos labai greitai. Eksperimentai parodė, kad tokie veiksmai priklausomai nuo skaičiavimų ir duomenų – trunka apie 60 ms. Kuo puslapyje daugiau funkcijų, kurios naudoja Ajax, tuo puslapis (kartu ir sistema) yra dinamiškesnė. 6 paveiksle pavaizduoto puslapio vidutinis atsidarymo laikas yra apie 133 ms. Vidutinis funkcijų vykdymo laikas yra apie 52 ms. Jei funkcijos nebūtų realizuotos sukurtu modeliu, vidutinis vykdymo laikas padidėtų apie 133 ms. Dėl puslapio greito pakartotino užkrovimo (naršyklės ir Tapestry karkaso dėka) puslapio atsidarymo laikas gali būti pakankamai mažas, bet eksperimentai parodė, kad jis yra sąlyginai didelis, žiūrint į Ajax funkcijų veikimo spartą.



7 pav. Žinučių keliavimo iki adresato laikai

Išvados

Išanalizavus situaciją pasaulyje buvo nustatyta:

- Web 2.0 atnešė naujas tradicijas ir reikalavimus internetinėms sistemoms ir puslapiams.
- Ajax tapo neatskiriama dinamiškos sistemos (puslapio) dalimi.
- Reikalinga esamų technologijų adaptacija dinamiškiems sistemų modeliams kurti.

Pasinaudojus esamomis ir įgytomis žiniomis buvo atlikta:

- Apibrėžtos dinamiškos sistemos.
- Sukurtas ir aprašytas bazinis dinamiškos sistemos (puslapio) Ajax modelis.
- Modelis realizuotas eksperimentinėje sistemoje.
- Įvertini eksperimentų rezultatai.

Įvertinti eksperimentų rezultatai parodė:

- Suprojektuotas modelis veikia.
- Pasirinktas serverio apklausinėjimo būdas nėra idealus, bet pakankamas naudojimui.
- Tapestry karkasas puikiai tinka dinamiškoms sistemoms kurti.
- Modelio realizacijoje apdorojamų komandų sąrašas yra lengvai išplečiamas tiek kliento, tiek serverio pusėse.
- Sukurta sistema yra dinamiška ir jos funkcijos, realizuotos su Ajax apytiksliai veikia sparčiau tiek, kiek užtrunka pačio puslapio atidarymas.
- Modelis lengvai yra išplečiamas ir gali būti patobulintas, nes modelis dar parodo bazinius Ajax veikimo principus ir nepaslepia jokios logikos, susijusios su Ajax po kitu karkasu (išskyrus tuos atvejus, kai nenaudojamas sukurtas modelis).

Literatūros sąrašas

[1] A. Mesbah ir A. van Deursen. “An Architectural Style for Ajax. IEEE Conference on Software Architecture (WICSA'07)”. 2007. Žiūrėta 2007-03-12. Prieiga per internetą http://ieeexplore.ieee.org/iel5/11118/35633/01690173.pdf?tp=&arnumber=1690173&isnumber=35633

[2] A. Verickas. ”Web 2.0 – naujoji interneto karta. Kompiuterija. PC World”. 2007 Sausio 15. Žiūrėta 2007-03-19. Prieiga per internetą http://www.kompiuterija.lt/zurnale/2007-01-15/web-20-%E2%80%93-naujoji-interneto-karta/

[3] J. James Garrett. “Ajax: A New Approach to Web Applications. // Adaptive Path.” 2005 Vasario 18. Žiūrėta 2005-12-03. Prieiga per internetą http://www.adaptivepath.com/publications/essays/archives/000385.php

[4] J. Stamey, T. Richardson. “Middleware Development with AJAX. //Journal of Computing Sciences in Colleges. 2006 Gruodis“. Žiūrėta 2007-03-19. Prieiga per internetą http://portal.acm.org/ft_gateway.cfm?id=1181948&type=pdf&coll=Portal&dl=GUIDE&CFID=14022313&CFTOKEN=82256928

[5] K. Smith. “Simplifying Ajax-Style Web Development. // IEE JNL, Web Technologies”. 2006 Gegužė. Žiūrėta 2007-02-14. Prieiga per internetą http://ieeexplore.ieee.org/iel5/2/34216/01631955.pdf?tp=&arnumber=1631955&isnumber=34216

[6] T. O`Reilly. “What Is Web 2.0. Design Patterns and Business Models for the Next Generation of Software“. 2005-09-30. Žiūrėta 2007-03-19. Prieiga per internetą http://www.oreillynet.com/pub/a/oreilly/tim/news/2005/09/30/what-is-web-20.html?page=1

[7] Wikipedia. “Web 2.0”. Žiūrėta 2007-03-19. Prieiga per internetą http://en.wikipedia.org/wiki/Web_2.0

[8] Wikipedia. “Ajax (programming)”. Žiūrėta 2007-03-20. Prieiga per internetą http://en.wikipedia.org/wiki/AJAX

Methodology of creating dynamic internet systems

This article is about the appearance of Web 2.0, its specific new technologies and changed requirements for internet based systems. In Web 2.0, demands for working with large portions of data (information) using simple internet browser are increasing rapidly. Therefore there is a goal to reach the same performance as desktop applications. Set of technologies, called Ajax (Asynchronous JavaScript and XML) is described in detail. Its purpose is to help creating dynamic internet based systems (web pages). Basic architecture model for creating dynamic system (page) is defined and created. This model is based on Spiar architecture and reveals basic Ajax usage fundamentals.