VILNIAUS UNIVERSITETAS
MATEMATIKOS IR INFORMATIKOS FAKULTETAS
PROGRAMŲ SISTEMŲ KATEDRA

# Asinchroninių metodų interneto technologijose tyrimas

## Analysis of Asynchronous Methods in Web Technologies

Magistro baigiamasis darbas

| | | |
|---|---|---|
| Atliko: | Justinas Rakita | (parašas) |
| Darbo vadovas: | Lekt. Tadas Savičius | (parašas) |
| Recenzentas: | Lekt. Mindaugas Žilinskas | (parašas) |

Vilnius – 2009

## Santrauka

Populiarėjant AJAX technologiją naudojančioms sistemoms vis labiau aktualios darosi šios technologijos įtakojamos problemos. Verta paminėti, kad kai kurie tradicinių internetinių sistemų trūkumai dar labiau paaštrėja ir yra sunkiau aptinkami naujos kartos internetinėse sistemose. Dėl šios priežasties dauguma programuotojų anksčiau ar vėliau susiduria su įprastomis AJAX aplinkos problemomis: naršyklės navigacijos mygtukų panaudojimu, adresų išsaugojimu, paieškos varikliukų problema ir pan.

Magistriniame darbe „Asinchroninių metodų interneto technologijose tyrimas“ pateikiamas AJAX pagrindu veikiančio karkaso projektas apima dažniausiai pasikartojančių problemų, kylančių kuriant šią technologiją naudojančias sistemas, sprendimus. Pateikiami sprendimai apima patį duomenų mainų mechanizmą, įskaitant patogesnio jo taikymo galimybes, navigacijos mygtukų panaudojimo problemas, suderinamumą su paieškos sistemų varikliukais, taip pat būdus kaip galima padidinti sistemos saugumą.

Raktiniai žodžiai: AJAX, karkasas, Java, Javascript.

## Summary

While internet applications based on AJAX technology becomes more and more popular, respectively problems rising from this technology becomes more and more serious. It is worth to mention, that some sorts of problems which comes from internet applications working in an old way, becomes more actual and harder to catch in systems of new generation. Due to this reason, lots of software developers sooner or later faces common AJAX problems: using of browser's buttons, bookmarking pages, search engines (like Google) problems, etc.

This work describes project of AJAX framework. Framework includes solutions for most common problems that faces programmers while developing AJAX based internet applications. Solutions covers the way of data exchange, including possibilities of using it more handy, browser's navigation button problems, compatibility with search engines. It also covers ways in which could be increased application's security.

Keywords: AJAX, framework, Java, Javascript.

**Turinys**

# 1 Įvadas

Magistriniame darbe „Asinchroninių metodų interneto technologijose tyrimas“ pateikiamas AJAX pagrindu veikiančio karkaso projektas. Darbas apima dažniausiai pasitaikančių problemų, kylančių kuriant naujos kartos internetines sistemas, sprendimus.

AJAX technologija suteikia žymiai daugiau galimybių programuotojams kuriant informacinių sistemų vartotojo sąsają. Dėl dinaminio tinklalapių atnaujinimo galimybės, kai užuot perkrovus puslapį atnaujinamas tik reikiamas jo fragmentas, bei plataus Javascript funkcionalumo, internetinės sistemos savo interaktyvumu tampa vis labiau artimos įprastoms darbastalio sistemoms (angl. *desktop applications*).

Tačiau populiarėjant AJAX technologiją naudojančioms sistemoms vis labiau aktualios darosi šios technologijos įtakojamos problemos. Verta paminėti, kad kai kurie tradicinių internetinių sistemų trūkumai dar labiau paaštrėja ir yra sunkiau aptinkami naujos kartos internetinėse sistemose. Dėl šios priežasties dauguma programuotojų anksčiau ar vėliau susiduria su įprastomis AJAX aplinkos problemomis: naršyklės navigacijos mygtukų panaudojimu, adresų išsaugojimu, paieškos varikliukų problema ir pan.

Kitas svarbus dalykas – pačio duomenų mainų mechanizmo realizacija. Nors gausu literatūros detaliai aprašančios mechanizmo įgyvendinimą, dauguma programuotojų renkasi karkasus jau turinčius šį funkcionalumą ir leidžiančius gana nesunkiai jį panaudoti. Didžiąją dalį tokių karkasų galima suskirstyti į dvi grupes. Vieni jų pateikia tik pačias pagrindines funkcijas ir didžiąją dalį problemų paliekantys programuotojams. Tuo tarpu kiti – priešingai, praktiškai nereikalauja Javascript žinių ir leidžia sąlyginai nesunkiai realizuoti vartotojo sąsają, tačiau yra ribotų galimybių ir sunkiai derinami su kitomis technologijomis.

Apžvelgus pagrindinius siūlomus karkasus (JSON-RPC, DWR, GWT ir ECHO2) galima daryti išvadą, kad pirmųjų dviejų karkasų taikymas reikalauja palyginti daug Javascript kalbos žinių, taip pat norint internetinę aplikaciją papildyti vaizdiniais komponentais arba reikės juos kurti patiems sistemos programuotojams, arba panaudoti kitų karkasų siūlomus komponentus, o tai neretai sukelia suderinamumo problemų (karkasų palyginimas pirmame priede). [Bra05] [Mer05]

Kiek kitokia situacija yra su likusiais dviem karkasais. Jų taikymui praktiškai nereikia ne tik Javascript kalbos, bet ir HTML žinių. Tačiau nors jų panaudojimas iš pirmo žvilgsnio atrodo optimaliausias būdas internetinėms sistemoms kurti, jis turi ir savų trūkumų. Tam tikrose situacijose karkasų teikiamo funkcionalumo gali nebeužtekti, o naujų karkasų, bibliotekų ar savo Javascrtipt kodo įtraukimas į internetinę programą, sukurtą panaudojant tokio tipo karkasus, gali

būti labai sudėtingas ar netgi neįmanomas (karkasų palyginimas antrame priede). [Mer06] [Ber07]

Galima daryti išvadą, kad magistriniame darbe kuriamas karkasas turėtų užpildyti spragą tarp šių dviejų tipų. Jis turėtų leisti vartotojui papildyti jau esamą funkcionalumą papildomomis funkcijomis, tačiau kartu pateikti dažniausiai naudojamas funkcijas bei vaizdinius komponentus.

Kuriant AJAX funkcionalumą realizuojantį karkasą, itin svarbu tinkamai parinkti jam papildomas funkcijas. Kai kurias jų, kaip pavyzdžiui įvestos informacijos korektiškumo tikrinimą, daugumos internetinių sistemų kūrėjai realizuoja savaip, nors jos ir atlieka praktiškai tą pačią funkciją. Tokiu atveju atsižvelgiant į literatūroje siūlomus bei praktikoje taikomus konkrečios problemos sprendimus galima pasirinkti geriausią variantą arba apjungti kelis ir gautą rezultatą įtraukti į kuriamą AJAX karkasą. Kitos papildomos funkcijos taikomos ganėtinai retai, todėl jų įtraukti nebūtina. Atskiras atvejis yra sudėtingesni vaizdiniai komponentai: kai kurios internetinės programos jų visai arba beveik visai nenaudoja, tuo tarpu kitos – jų naudoja ganėtinai daug. Kaip vienas iš problemos sprendimo būdų galėtų būti atskira vaizdinių komponentų funkcionalumą realizuojanti karkaso dalis, kurios įtraukimą į internetinę programą lemtų vartotojo pasirinkimas.

Verta paminėti, kad vienas iš svarbiausių internetinės sistemos, o tuo pačiu ir jos sudėtyje esančio AJAX funkcionalumą realizuojančio karkaso kriterijų yra informacijos saugumas. Kuriant karkasą svarbu atsižvelgti į esamas problemas bei taikyti jų sprendimus, pavyzdžiui užšifruoti kliento pusėje veikiantį Javascript kalbos kodą, kad sistemos vartotojas negalėtų jo perskaityti ir panaudoti blogiems tikslams.

Apžvelgus literatūrą bei publikacijas internete buvo sudarytas problemų, kurias turėtų spręsti karkasas, sąrašas:

Pirmoji problemų grupė – trūkumai pasikartojantys didžiojoje dalyje šaltinių: naršyklės navigacijos mygtukų veikimas, adresų saugojimas ir suderinamumas su paieškos varikliukais.

Antroji grupė – saugumo problemos. Grupė sudaryta atsižvelgiant į *Ajax security* ([HofSul07]) knygos autorių pastabas. Pasirinktos tos problemos, kurias galima išspręsti realizuojant vieną ar kitą karkaso funkcionalumą.

Trečiąją grupę sudaro problemos, kurios yra aktualios ir tradicinėms, nenaudojančioms AJAX mechanizmo, internetinėms sistemoms. Tai yra duomenų mainų inicijavimas serverio pusėje (pasirinktas atsižvelgiant į *Echo2* karkaso architektūrą) bei nenumatytų situacijų apdorojimas.

Ketvirtoji grupė apima būsenos saugojimo ypatumus. Natūralu, kad daliai verslo logikos persikėlus į kliento pusėje veikiančią sistemos dalį atsirado poreikis šioje pusėje esančią informaciją saugoti bei sinchronizuoti su serverio pusėje esančia dalimi.

Magistrinio darbo „Asinchroninių metodų interneto technologijose tyrimas“ tikslas – išnagrinėjus technologijų, realizuojančių nuotolinį metodų kvietimą, privalumus bei trūkumus rasti ir pateikti būdus pastebėtiems trūkumams šalinti, apjungiant juos į karkaso projektą.

Keliami uždaviniai:

- Identifikuoti dažniausiai pasitaikančias ir labiausiai aktualias problemas;
- Rasti jų sprendimo būdus
- Pagal rastus variantus ir savo idėjas, kiekvienai problemai rasti optimalų sprendimą.

## 2 Karkaso branduolys

Vienas iš svarbiausių dalykų AJAX pagrindų veikiančiose internetinėse sistemose yra XmlHttpRequest užklausos realizacija, skirta duomenų mainams tarp kliento ir serverio pusių. Atitinkamai beveik visi egzistuojantys AJAX karkasai siūlo vienokį ar kitokį sprendimą šiai funkcijai atlikti, t.y. pagal kliento pusėje esančią informaciją suformuoti duomenų paketą, nusiųsti jį serverio pusėje veikiančiai sistemos daliai bei gautą atsakymą paversti Javascript kalbai suprantamais duomenimis ir pagal juos atlikti reikiamus veiksmus.

Esant dideliam duomenų mainų realizacijos variantų pasirinkimui, nėra tikslinga kurti dar vieną sprendimą: tai užimtų nemažai laiko, be to paliktų galimybę atsirasti klaidoms, kurių žymiai mažiau pasitaiko seniau naudojamuose ir tuo pačiu gerai išbandytuose sprendimuose. Apžvelgus egzistuojančius karkasus detalesniam nagrinėjimui buvo pasirinkti du iš jų: DWR (angl. *Direct Web Remote*) ir JSON-Java. Karkasai duomenų perdavimui naudoja skirtingas technologijas. DWR duomenis perduoda XML formatu ir naudoja XML-RPC protokolą, tuo tarpu JSON-Java informaciją perduoda JSON (angl. *JavaScript Object Notation*) formatu, naudodamas JSON-RPC protokolą. Abu šie sprendimai yra gana plačiai naudojami bei pasižymi siūlomų galimybių gausa, tačiau DWR yra patogesnis naudoti ir turi gausesnį funkcionalumą negu JSON-Java.

DWR yra atviro kodo technologija (taikoma Apache licenzija), kurios principas remiasi tuo, kad Java objektuose esantys metodai gali būti kviečiami tiesiai iš naršyklėje vykdomo JavaScript. Technologija internetinėje programoje atlieka XMLHttpRequest užklausų-atsakymų ciklą bei duomenų vertimą į XML formatą ir atgal.

Naudojant DWR, Java metodams galima perduoti ir iš jų priimti įvairių tipų parametrus, įskaitant tokius kaip masyvai, kolekcijos, objektai bei DOM (angl. *Dynamic Object Model*). Verta paminėti ir tai, kad naudojant šį sprendimą HTML išeities tekstas (angl. *source*) išlieka lengvai skaitomas. DWR nėra susietas su jokiu karkasu, todėl jis veikia su visais pagrindiniais J2EE internetinių programų konteineriais (angl. *container*).

Technologiją sudaro dvi dalys:

1. Serverio pusėje patalpinta DWR serverinė programa (angl. *servlet*), kuris priima ir apdoroja gaunamas užklausas ir siunčia atsakymus atgal naršyklei;
2. Kliento pusėje vykdomas JavaScript kodas, kuris siunčia užklausas ir priklausomai nuo gautų atsakymų gali dinamiškai atnaujinti tinklalapį.

DWR dinamiškai generuoja JavaScript kodą, priklausomai nuo Java klasių, kurių metodai bus kviečiami. Kiekvienai klasei sugeneruojamas atskiras kodo fragmentas. Reikalingi metodai ir jų klasės nurodomi dwr.xml faile, kuris turi būti patalpintas WEB-INF kataloge. Sugeneruotas

kodas atlieka duomenų mainus tarp kliento ir serverio pusių, nors dėl užslėpto XMLHttpRequest užklausų kvietimo susidaro įspūdis, kad visos operacijos atliekamos naršyklėje.

Nepaisant to, kad DWR leidžia pakankamai patogiai realizuoti duomenų mainus tarp kliento ir serverio pusių bei atsižvelgia į kai kurias saugumo problemas, lieka pakankamai nemažai neišspręstų problemų būdingų AJAX pagrindu veikiančioms sistemoms.

## 2.1 Įvykių ir atsinaujinančių objektų susiejimas

Ganėtinai svarbus dalykas kuriant AJAX pagrindu veikiančias sistemas – galimybė nesudėtingai taikyti dinaminį tinklalapių turinio atnaujinimą. Nors DWR leidžia patogiai realizuoti duomenų mainus tarp kliento ir serverio pusių, tiek asinchroninės užklausos inicijavimas, tiek puslapio atnaujinimas pagal gautus duomenis yra paliekami programuotojui. Kai kuriais atvejais tai yra naudinga, nes programuotojas gali laisvai ir be jokių apribojimų įgyvendinti norimą funkcionalumą, arba savo nuožiūra taikyti darbui su vartotojo sąsaja skirtas Javascript bibliotekas. Tačiau kur kas dažniau, tai sukelia papildomų problemų. Tai ypač aktualu mažiau patirties turintiems programuotojams. Jeigu vartotojo sąsajos realizavimui nenaudojamos jokios papildomos bibliotekos ar karkasai, toks darbas ne tik užima žymiai daugiau laiko bei reikalauja pakankamai gilių Javascript, o kartais ir CSS (angl. *Cascade Style Sheets*) žinių, bet ir gerokai padidina klaidų bei naršyklių nesuderinamumo tikimybę.

Planuojamame AJAX karkase numatomi trys vartotojo sąsajos realizavimo variantai:

1. Galimybė paprastai susieti dažniau naudojamus komponentus (pvz. mygtuką ir teksto įvedimo lauką);
2. Sudėtingesni, bet pakankamai dažnai naudojami komponentai;
3. Galimybė programuotojui savarankiškai (arba naudojant pasirinktą Javascript biblioteką) realizuoti vartotojo sąsają;

### 2.1.1 Komponentų susiejimas

Pakankamai daug internetinių sistemų AJAX funkcionalumą naudoja tik paprastiems veiksmams įgyvendinti, pvz. pasirinkus punktą iš reikšmių lauko, atnaujinamas šalia esantis reikšmių laukas. Tokiais atvejais kliento pusėje paprastai nebūna kokių nors skaičiavimų ar kitų operacijų atlikimo. Tiesiog inicijavus tam tikrą veiksmą, turi būti atnaujinama atitinkama tinklalapio sritis. Žinoma, beveik visada tai reiškia tam tikrų operacijų atlikimą serverio pusėje, pavyzdžiui, naujos informacijos iš duomenų bazės nuskaitymą, tačiau tai dažniausiai nesukelia problemų.

Kiek kitokia situacija su tokių sprendimų realizavimu kliento pusėje. Norėdamas įgyvendinti panašų funkcionalumą programuotojas turi:

1. Aprašyti Javascript funkciją, kuri bus kviečiama inicijuojant tam tikrą veiksmą, pvz. mygtuko paspaudimą. Funkcija turi realizuoti ar kitaip inicijuoti AJAX užklausos kvietimą;
2. Priskirti aprašytą funkciją norimam komponentui (pvz. reikšmių sąrašui);
3. Aprašyti funkciją, kuri būtų kviečiama grįžus užklausos rezultatui. Funkcijoje turėtų būti apdorojamas gautas atsakymas bei atliekami reikiami pakeitimai DOM objekte, tuo pačiu dinamiškai atnaujinant tinklalapį.

Kadangi praktiškai visos minėtos užduotys turi būti realizuojamos Javascript kalba, atitinkamai iškyla poreikis palyginti išsamaus šios kalbos išmanymo, netgi tais atvejais, kai realizuojami funkcionalumai pakankamai primityvūs. Tačiau net ir turint neblogus programavimo Javascript kalba įgūdžius, kiekvienas toks komponentų susiejimas reikalauja laiko. Taip pat negalime pamiršti ir naršyklių suderinamumo problemos, kuri kalbant apie DOM objekto redagavimą tampa ypač aktuali. Atitinkamai iškyla poreikis paprasto, bet universalaus sprendimo tokioms situacijoms.

#### 2.1.1.1 Sprendimo realizacija

##### 2.1.1.1.1 Iniciatoriaus aprašymas

Veiksmo iniciatoriui (mygtukui, reikšmių pasirinkimo laukui ar pan.) turi būti nurodoma, kokia funkcija turės būti iškviesta inicijavus veiksmą, kokie parametrai, jai turės būti perduodami bei koks komponentas turės būti atnaujintas atlikus operacijas serverio pusėje. Tai atliekama mygtukui ar kitam veiksmą inicijuojančiam komponentui pridedant tam skirtus atributus ir jiems priskiriant atitinkamas reikšmes (žiūrėti trečią priedą).

Pirmuoju atveju bus atnaujinamas tik vienas nurodyto elemento atributas. Atributui *resultField* turi būti priskirta elemento, kuris bus atnaujintas, identifikatoriaus reikšmė. Atributui *parameterFields* turi būti priskirti identifikatoriai tų laukų, kurių reikšmės vėliau bus naudojamos operacijoms atlikti. Jeigu tokių laukų yra daugiau negu vienas, identifikatoriai turi būti atskirti tarpais. Taip pat labai svarbu užtikrinti, kad identifikatoriai bus pateikti reikiama tvarka. Atributui *resultAttribute* turi būti nurodyta, koks atnaujinamo elemento atributas įgaus naują reikšmę, jeigu *resultAttribute* nėra aprašytas, nauja reikšmė turi būti priskirta atributui *value*. Atributui *methodToCall* turi būti nurodytas Java metodo, kurį reikia iškviesti, pavadinimas.

Antruoju atveju priklausomai nuo nustatymų nurodytas elementas bus pakeistas elementu, kuris bus gautas kaip rezultatas arba į nurodytą elementą bus įterptas gautas rezultatas kaip vidinis elementas. Šiuo atveju vietoje *resultAttribute* atributo turi būti nurodytas *setValue*

atributas, kuris gali įgyti vieną iš dviejų reikšmių: *insert* – gauto rezultato įterpimui arba *replace* tam atvejui, kai gautas rezultatas įterpiamas kaip vidinis elementas.

Kadangi minėti laukų atributai nėra numatyti W3C standarte, jų aprašymas turi būti įtrauktas į DTD failą.

#### 2.1.1.1.2 Funkcijos aprašymas

Inicijavus veiksmą (paspaudus mygtuką, pasirinkus reikšmę iš sąrašo ar pan.), turi būti kviečiama karkaso funkcija, kuriai perduodamas veiksmą inicijavusio komponento objektas. Pagal nuskaitytus objekto atributus kviečiama DWR pagrindu veikianti funkcija, kuri savo ruožtu iškvies Java metodą. Funkcijos kvietimui reikalingi parametrai:

- Metodo pavadinimas. Parametras nusako, kokį Java metodą reikia kviesti;
- Parametrų reikšmės. Parametrą sudaro masyvas, su vartotojo įvestais duomenimis, reikalingais rezultatui gauti;
- Rezultatų elemento identifikatorius. Parametras nurodo, koks laukas turės būti koreguojamas grąžinus rezultatą.
- Nustatymai. Parametrą sudaro masyvas, kuriame laikoma informacija apie rezultato pateikimą: ar tai bus DOM fragmentas, ar tiesiog vieno atributo reikšmė. Jeigu tai bus DOM fragmentas ar jis turi pakeisti ankstesnį elementą, ar jį papildyti. Jeigu tai bus atributo reikšmė, kokiam atributui ją reikia priskirti.

#### 2.1.1.1.3 Serverinės dalies aprašymas

Kiekviena DWR pagrindu veikianti funkcija turi kviesti Java metodą *manageRequest* esantį *RequestBackingBean* klasėje. Metodas turi:

- Patikrinti gautų parametrų korektiškumą bei aptikus netikslumų ar trūkumų grąžinti pranešimą apie klaidą;
- Jeigu paaiškėja, kad gauti parametrai korektiški, iškviesti metodą pagal parametrą *methodToCall*.
- Grąžinti gautą rezultatą klientinei pusei, kviečiant grįžtamąją funkciją (*callback*).

Nurodytą metodą aprašo pats vartotojas, jis gali grąžinti arba reikšmę, kuri vėliau bus priskirta reikiamam atributui, arba DOM fragmentą.

#### 2.1.1.1.4 Rezultatų apdorojimas kliento pusėje

Rezultatus apdorojančiai Javascript funkcijai kaip parametrai turi būti pateikiami gautas rezultatas, bei anksčiau suformuotas nustatymų masyvas. Priklausomai nuo nustatymų masyvo

reikšmių, gautas rezultatas yra priskiriamas atitinkamam atributui arba įterpiamas į tam tikrą DOM objekto vietą.

Sprendimo realizacija pateikta ketvirtame priede.

1. Veiksmo iniciatoriui (mygtukui, reikšmių pasirinkimo laukui ar pan.) turi būti nurodomas identifikatorius tinklalapio komponento, kurį reikės atnaujinti. Identifikatorius turėtų būti nurodomas per tam skirtą komponento atributą;
2. Inicijavus veiksmą (paspaudus mygtuką, pasirinkus reikšmę iš sąrašo ar pan.), turi būti kviečiama karkaso funkcija, kuriai perduodamas atnaujinamo lauko identifikatorius, bei nauja pasirinkta ar įvesta reikšmė (jeigu tokia yra);
3. Karkaso funkcija panaudodama DWR funkcionalumą turi kviesti bendrą elementų atnaujinimui skirtą Java kalbos metodą ir jam perduoti:
   a. naują reikšmę (jeigu tokia yra);
   b. lauko, kurį reikia atnaujinti, identifikatorių;
   c. elemento, inicijavusio veiksmą, identifikatorių;
   d. jeigu yra poreikis ir DOM objekto fragmentą, aprašantį lauką kurį reikia atnaujinti;
4. Java metodas pagal gautus lauko bei elemento identifikatorius turi iškviesti atitinkamą konkretaus lauko atnaujinimui skirtą metodą ir perduoti jam visus gautus duomenis;
5. Atitinkamame Java metode pagal gautus duomenis turi būti suformuojamas DOM objekto fragmentas (gali būti tiesiog paredaguojamas gautas elemento DOM objektas);
6. Suformuotas rezultatas turi būti grąžinamas kliento pusėje veikiančiai internetinės sistemos daliai ir pagal atnaujinamo lauko ID į atitinkamą vietą įterpiamas gautas DOM objekto fragmentas (galimas variantas, kai DOM objekto fragmentas įterpiamas vietoje ankstesnio elemento).

## 2.2 Sudėtingesni komponentai

Daugeliu atvejų kuriant internetines sistemas prisireikia kiek sudėtingesnių vartotojo sąsajos elementų, kuriuose taip pat turi būti realizuojamos AJAX pagrindu veikiančios užklausos. Žinoma, yra pakankamai daug bibliotekų, siūlančių tokio pobūdžio vartotojo sąsajos komponentus, tačiau verta paminėti, kad jose paprastai būna realizuoti tik patys vaizdiniai komponentai, o duomenų atnaujinimo ar perdavimo funkcijos paliekamos realizuoti programuotojams.

### 2.2.1 Sprendimo realizacija

Į karkaso modelį turi būti įtraukti dažniau naudojami vartotojo sąsajos komponentai. Komponentai turi būti gaunami praplečiant *dojo*, *MooTools* ir jeigu reikia kitų karkasų komponentų funkcionalumus. Į karkasą turi būti įtraukti šie komponentai:

#### 2.2.1.1 Duomenų lentelės

Pagrindiniai reikalavimai duomenų lentelėms yra dinaminis rūšiavimas, puslapiavimas bei užslėptas gretimų puslapių užkrovimas.

Rūšiavimas turi būti realizuotas serverio pusėje, nes esant dideliam duomenų kiekiui nėra tikslinga jų visų siųsti į klientinę sistemos dalį, o įvykdžius rūšiavimą tik kliento pusėje esantiems duomenims rezultatai neatitiktų tikrovės. Duomenų lentelės klientinė dalis turi būti realizuojama panaudojant *dojo* karkaso komponentą. Kadangi iš anksto nėra žinoma kaip bus formuojama užklausa bei kokia duomenų bazė bus naudojama, serverinėje dalyje turi būti realizuoti metodai:

- *getTableForData(String queryName)* – pagal gautą užklausos pavadinimą (query name) turi būti kviečiamas vartotojo aprašytas metodas;
- *getTableForData(String queryName, String sortBy, String sortType)* – skirtingai nuo pirmojo, šis metodas turi du papildomus parametrus, kuriais nusakoma pagal kurį stulpelį turi būti rūšiuojami rezultatai (parametras sortBy) bei kokia tvarka (parametras sortType, galintis įgyti reikšmes ASC – rūšiavimui didėjimo tvarka arba DESC – rūšiavimui mažėjimo tvarka).

Puslapiavimo realizavimui vartotojo sąsaja turi būti papildyta pasirinkimų sąrašu, leidžiančiu nurodyti kiek reikšmių turi būti atvaizduojama, bei nuorodomis, reikalingomis naviguoti per lentelės puslapius. Serverinėje dalyje anksčiau minėti metodai *getTableForData(String queryName)* ir *getTableForData(String queryName, String sortBy, String sortType)* turi būti papildyti parametrais *from* ir *to*, nusakančiais, koks užklausos rezultatų intervalas turi būti atvaizduojamas. Parametrų *from* ir *to* reikšmės turi būti apskaičiuojamos klientinėje sistemos dalyje pagal pasirinktą lentelės puslapį ir reikšmių, atvaizduojamų viename puslapyje, kiekį (t.y. jeigu yra atvaizduojamas trečias lentelės puslapis, o viename puslapyje yra pateikiamos 25 reikšmės, parametrų *from* ir *to* reikšmės bus atitinkamai 51 ir 75).

Klientinėje sistemos dalyje turi būti realizuotas funkcionalumas, leidžiantis pagreitinti navigaciją per puslapius, užkraunant gretimus puslapius iš anksto. Tai turi būti atliekama klientinėje sistemos dalyje aprašant du objektus: mažesnį ir didesnį indeksus turinčių puslapių turinio saugojimui. Pirmą kartą užkrovus lentelę, iš karto ją atvaizdavus į vieną iš objektų turi būti užkraunamas gretimo puslapio turinys. Kadangi paprastai tik užkrovus lentelę būna

atvaizduojamas pirmasis puslapis, iškart po atvaizdavimo turi būti nuskaitomas antrojo puslapio turinys. Vartotojui pasirinkus gretimą puslapį, esamas turinys turi būti talpinamas į vieną iš turinio saugojimui skirtų kintamųjų, kito kintamojo turinys turi būti atvaizduojamas, o į jo vietą užkraunamas gretimo puslapio turinys. T.y. jeigu yra atvaizduojamas pirmasis puslapis, viename iš kintamųjų jau būna saugojamas antrojo puslapio turinys. Vartotojui naviguojant į antrąjį, pirmojo puslapio turinys išsaugomas kintamajame, į kurį talpinamas mažesnį indeksą turinčio puslapio turinys, iš kito kintamojo paimamas ir atvaizduojamas antrojo puslapio turinys, o jį atvaizdavus nuskaitomas gretimo – trečiojo puslapio turinys. Duomenų lentelės pavyzdys atvaizduotas pirmame paveiksle.

| Vardas | Pavardė | Gimimo metai | |
|---|---|---|---|
| Jonas | Jonaitis | 1984 | Šalinti |
| Petras | Petraitis | 1973 | Šalinti |
| Vardenis | Pavardenis | 1961 | Šalinti |
| | | | Pridėti |

*1 pav. Duomenų lentelė*

#### 2.2.1.2 Dinaminis sąrašas

Pagrindiniai reikalavimai dinaminio sąrašo elementui yra galimybė neperkraunant puslapio įtraukti naujas reikšmes bei redaguoti ar šalinti jau įvestas reikšmes. Dinaminį sąrašą sudaro lentelė, kurios ląstelėse yra teksto įvedimo laukai. Priklausomai nuo nustatymų gali būti naudojami tik skaitymui skirti laukai, kurie, ant jų paspaudus pele, virsta teksto įvedimo laukais. Pakoregavus tokio lauko reikšmę, ir pasirinkus kitą lauką arba spragtelėjus pele už lentelės ribų jis vėl taptų tik skaitymui skirtu lauku, o nauji duomenys būtų automatiškai išsaugoti. Kiekvienos eilutės, išskyrus pačią paskutinę, gale yra šalinimui skirtas mygtukas, kurį paspaudus eilutė pašalinama. Prieš šalinant eilutę turi būti atvaizduotas pranešimas „Ar tikrai norite eilutę pašalinti? Taip/Ne“. Paskutinė lentelės eilutė turi būti tuščia ir skirta naujiems duomenims įvesti, t.y. visi jos laukai turi būti tušti ir jos gale turi būti duomenų saugojimui skirtas mygtukas.

AJAX pagrindu veikianti užklausa turi būti kviečiama kiekvieną kartą, kai:

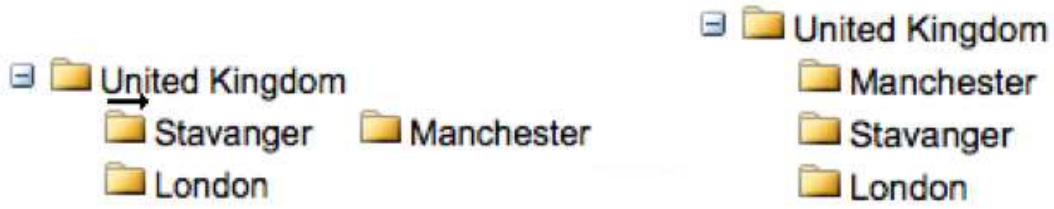
- Pašalinama duomenų eilutė. Paspaudžiamas eilutės šalinimui skirtas mygtukas ir patvirtinimo lange pasirenkama reikšmė „Taip“ (šalinti);
- Pakoreguojama lentelės lauko reikšmė ir pasirenkamas kitas laukas ar kitu būdu žymeklis perkeliamas į kitą vietą;
- Paspaudžiamas naujos eilutės pridėjimui skirtas mygtukas. Šiuo atveju prieš kviečiant užklausą reikia patikrinti ar bent vienas iš laukų yra užpildytas.

Serverinėje sistemos dalyje dinaminio sąrašo atitikmuo būtų *java.util.List* tipo sąrašas, kurio elementai būtų *String* tipo eilučių sąrašai. Pirmasis sąrašas atitiktų visą lentelę, antrasis – vieną lentelės eilutę.

### 2.2.1.3 Medžio pavidalo sąrašas

Karkaso sudėtyje turi būti numatytas elementas pasižymintis visomis anksčiau minėto dinaminio sąrašo savybėmis, tačiau turintis medžio tipo struktūrą. Toks elementas privalo pasižymėti šiomis savybėmis:

- turi būti numatyta galimybė dinamiškai įtraukti naują sąrašo elementą (šaką);
- turi būti numatyta galimybė dinamiškai pašalinti jau egzistuojantį sąrašo elementą (šaką), kartu su visais žemesnio lygmens elementais;
- turi būti numatyta galimybė dinamiškai perkelti jau egzistuojantį sąrašo elementą (šaką), kartu su visais žemesnio lygmens elementais. Perkėlimas turi būti realizuojamas taip, kad priklausomai nuo to kur „paleidžiamas“ perkeliamas elementas jis gali tapti arba to paties lygmens elementu kaip ir tas į kurį rodo pelės žymeklis paleidimo momentu, arba žemesnio lygmens elementu;




*2 pav. Perkėlimas įtraukiant kaip žemesnio lygmens elementą.*


*3 pav. Perkėlimas įtraukiant kaip to paties lygmens elementą.*

- turi būti numatyta galimybės dinamiškai išskleisti ir paslėpti žemesnio lygmens elementus;

- turi būti numatyta galimybė užkrauti tik išskleistas šakas. Toks apribojimas turėtų įtakos esant didelės apimties medžio pavidalo struktūroms.

Klientinė dalis turi būti realizuota panaudojant *dhtmlgoodies* bibliotekos komponentą *Folder tree with drag and* drop ir išplečiant jo savybes. Serverio pusėje veikianti sistemos dalis turi būti realizuota panaudojant ArrayList tipo sąrašą, kurio elementai būtų medžio šakos. Sąrašo elementai turi būti objektai, kuriuose laikoma elemento pavadinimas, identifikatorius bei tėvinio elemento identifikatorius. Kiekvieną kartą atliekant keitimus kliento pusėje turi būti kviečiama užklausą serverio pusei, inicijuojanti informacijos apie pakeitimus perdavimą.

Dinaminiam naujo elemento įtraukimui ar jau egzistuojančio elemento šalinimui turi būti panaudojamas komponento meniu. Jame turi būti naujos šakos ir šiukšlių dėžės piktogramos. Norint įtraukti naują elementą reikia pele „paimti“ už naujos šakos piktogramos ir ją „nutempti“ į norimą vietą. Šiuo atveju svarbu paminėti kelis dalykus: pele „paėmus“ už naujos šakos piktogramos, jos objektas DOM modelyje turi būti klonuojamas, t.y. sukuriamas antras lygiai toks pats elementas vietoje „paimto“, kitas dalykas – „padėjus“ naują elementą į norimą vietą, jo pavadinimas turi tapti redaguojamas, t.y. iš tik skaitymui skirto lauko jis turi virsti teksto įvedimo lauku.

Elemento šalinimas turi būti atliekamas „užtempiant“ jį ant šiukšlių dėžės piktogramos. Prieš vykdant šalinimą turi būti išvedamas pranešimas, papildomai paklausiantis ar sistemos vartotojas tikrai nori pašalinti elementą ir visus jo vaikus. Pasirinkus neigiamą atsakymą, elementas turi būti grąžinamas į ankstesnę vietą. Šalinant elementus serverio pusėje turi būti rekursyviai einama per elementų sąrašą, iš jo išmetant nurodytą elementą bei visus jo vaikus.

Perkeliant elementą į naują poziciją, serverio pusėje užtenka pakeisti atitinkamo elemento lauką, nurodantį, kokiam tėviniam elementui jis priklauso. Žemesnio lygmens elementų laukai nesikeičia.

#### 2.2.1.4 Puslapio „rakinimas“

Karkaso sudėtyje turi būti numatytas pranešimo apie vykstantį puslapio atnaujinimą atvaizdavimas, tuo pačiu padarant likusią puslapio dalį nepasiekiama (ją „užrakinant“). Toks funkcionalumas reikalingas dėl kelių priežasčių: reikia pranešti sistemos vartotojui apie vykstančius pasikeitimus, o tuo pačiu neretai reikia apsaugoti sistemą nuo nepageidaujamų vartotojo veiksmų atnaujinimo metu. Taip pat turi būti realizuojami ir alert tipo pranešimai, suteikiant galimybę derinti jų stilių ir turinį.

#### 2.2.1.5 Sprendimo realizacija

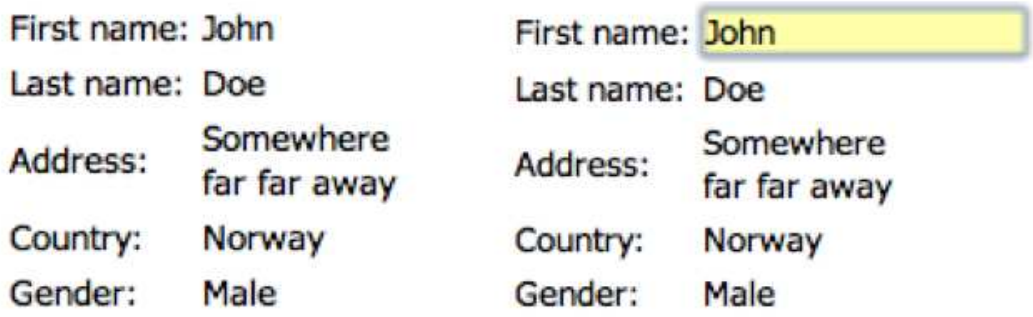
Šios karkaso savybės turi būti realizuojami panaudojant CSS funkcionalumą, leidžiantį išdėstyti elementus keliais sluoksniais (*z-index*). Pranešimas turi būti atvaizduojamas aukštesniame sluoksnyje, taip neleidžiant sistemos vartotojui pasiekti žemiau esančių elementų. Turi būti realizuotos dvi Javascript funkcijos: viena pranešimo atvaizdavimui, kita atvaizdavimo panaikinimui;

### 2.2.2 Redaguojamas-neredaguojamas teksto laukas

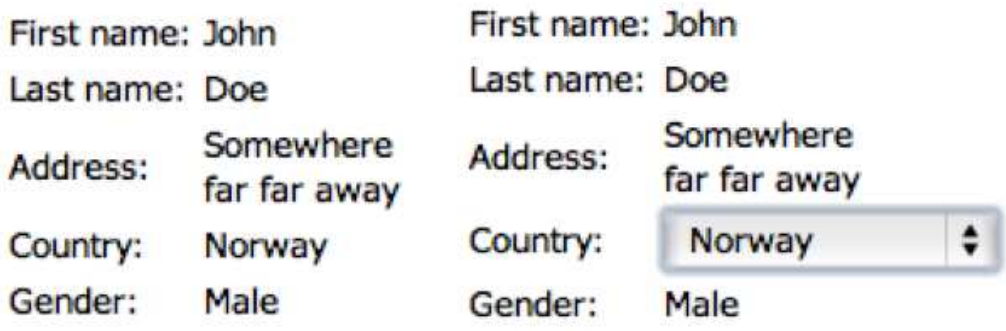
Į karkasą turi būti įtrauktas tik skaitymui skirtas laukas, kuris paspaudus pele tampa redaguojamas. Toks komponentas turi būti naudojamas ir duomenų lentelėje, jeigu yra poreikis dinamiškai redaguoti jos duomenis. Tokiu atveju visi jos laukai galėtų būti redaguojami, o tuo pačiu nebūtų iškraipomas lentelės stilius, pildant ją teksto įvedimo laukais;

#### 2.2.2.1 Sprendimo realizacija

Funkcionalumui realizuoti turi būti panaudotas anksčiau minėtos dhtmlgoodies bibliotekos komponentas, leidžiantis atvaizduoti tekstą, kuris jį spragtelėjus virsta reikšmių įvedimo ar pasirinkimo lauku.




*4 pav. Reikšmių įvedimo laukas.*




*5 pav. Reikšmių pasirinkimo laukas.*

Funkcionalumas būtų aprašomas panaudojant CSS technologiją (žiūrėti penktą priedą).

### 2.2.3 Vietinis meniu

Karkase turi būti numatytas vietinis meniu, iškviečiamas pelės mygtuko paspaudimu. Komponentas turi būti realizuotas greta pelės žymeklio atvaizduojant (pakeičiant atributą iš *display:none* į *display:block*) lentelę su nuorodomis arba mygtukais. Pvz., paspaudus ant lentelės elemento vietiniame meniu galėtų būti pasirinkimai: „Redaguoti elementą“, „šalinti eilutę“ arba „šalinti stulpelį“.

#### 2.2.3.1 Sprendimo realizacija


*6 pav. Vietinis meniu.*

Meniu aprašomas panaudojant sąrašui formuoti skirtas žymas (žiūrėti šeštą priedą). Atvaizduojamas CSS žymą *display:none* keičiant į *display:block*. Pozicija nustatoma pagal *event* objekto *clientX* ir *clientY* žymas.

## 2.3 Tiesioginis DWR naudojimas

Greta pateikiamų komponentų bei galimybės aprašyti tik tam tikrų tinklalapio elementų susiejimą, turi būti paliekama galimybė programuotojui naudoti pasirinktą Javascript biblioteką vartotojo sąsajai realizuoti bei panaudojant DWR siūlomą funkcionalumą, realizuoti AJAX pagrindu veikiančių užklausų kvietimą.

## 2.4 Rezultatai

Skyriuje „Karkaso branduolys“ aprašomos pačios pagrindinės karkaso funkcijos: duomenų perdavimo mechanizmas, bei dinaminio tinklalapio komponentų atnaujinimo galimybės.

Pats duomenų perdavimo mechanizmas realizuojamas panaudojant DWR karkasą. Šitas variantas pasirinktas kaip labiausiai tinkamas, dėl patogaus naudojimo, suderinamumo su kitomis technologijomis, bei galimybės pačiam programuotojui išplėsti jo funkcionalumą.

Pagrindinis minėto karkaso trūkumas yra tai, kad jį naudojant programuotojas pats turi realizuoti vartotojo sąsają, įskaitant ir įvykių apdorojimą. Ši problema sprendžiama realizuojant mechanizmą leidžiantį nesunkiai susieti elementą į kurio įvykius reikia reaguoti ir elementą,

kuris reaguojant į minėtus įvykius turėtų būti atnaujinamas. Pvz. paspaudus mygtuką teksto laukas įgauna naują reikšmę.

Greta šio funkcionalumo yra pateikiama ir grupė dažniausiai naudojamų komponentų, kurių realizacija yra sudėtingesnė (duomenų lentelė, dinaminis sąrašas ir pan.).

# 3 Navigacijos problemos

Vienos iš labiausiai žinomų problemų, su kuriomis susiduria programuotojai naudojantys AJAX technologiją, kyla dėl pasikeitusio informacijos atnaujinimo principo. Tradicinėse internetinėse sistemose vartotojui paspaudus ant nuorodos arba mygtuko, būdavo atsiunčiamas ir atvaizduojamas naujas tinklalapis. Praktiškai kiekvienas toks tinklalapis turėdavo nuorodą, kuria pasinaudojus jis galėdavo būti pasiekiamas. Tokiu būdu tinklalapio adresas be didesnių problemų galėdavo būti išsaugomas naršyklės adresyne, taip pat tokiomis nuorodomis galėdavo nesunkiai pasinaudoti ir internetinių paieškos sistemų varikliukai.

AJAX pagrindu veikiančiose internetinėse sistemose atlikus vieną ar kitą veiksmą paprastai būna atnaujinamas tik tinklalapio fragmentas ir nuoroda į tą tinklalapį nepasikeičia. Tokiu atveju gali būti išsaugomas tik pradinis sistemos puslapis. Kita problema yra tai, kad paieškos sistemų varikliukai nesugeba naviguoti per sistemą, kurioje realizuotas dinaminis tinklalapio atnaujinimas.

## 3.1 Mygtukas „Grįžti“ bei adresų saugojimas

Viena iš AJAX technologijos problemų yra navigacijos mygtukų naudojimas. Kai kuriose AJAX pagrindu veikiančiose internetinėse sistemose naršyklėse esantys navigacijos mygtukai neveikia arba veikia ne taip kaip vartotojas tikisi.

Navigacijos mygtukų paskirtis yra atverti puslapius, kurie pagal naršymo istoriją buvo atverti prieš arba po einamojo puslapio. Tradicinėse internetinėse sistemose vartotojui paspaudus nuorodą, yra siunčiama užklausa serverinei sistemos daliai ir pagal gautus duomenis suformuojamas naujas puslapis. Atitinkamai naršyklės istorijoje atsiranda naujas įrašas, atitinkantis užkrautą puslapį.

Tuo tarpu AJAX pagrindu veikiančiose sistemose gana dažnai nauji puslapiai nėra kuriami. Vietoje to, pagal asinchroninės užklausos rezultatą yra atnaujinamas tam tikras puslapio fragmentas. Kadangi nesukuriamas naujas puslapis, atitinkamai ir naršyklės istorijoje neatsiranda jokio naujo įrašo. Tokiais atvejais, sistemos vartotojui paspaudus mygtuką „Grįžti“ yra tiesiog grįžtama į paskutinį prieš tai užkrautą puslapį, o tai reiškia, kad kai kuriais atvejais netgi išeinama už naudojamos internetinės sistemos ribų.

### 3.1.1 Galimi sprendimų variantai

Yra keletas galimų variantų problemai spręsti. Vienas jų DOM (angl. *Dynamic Object Model*) esančio objekto *window* ir jo savybės *location* panaudojimas. Tačiau šis variantas gali būti taikomas tik *Firefox* bei *Opera* naršyklėse. Internet Explorer naršyklėje dėl savito podėliavimo (angl. *caching*) problema išliks, o kadangi kaip tik šią naršyklę naudoja didžioji dalis interneto vartotojų, toks sprendimas netinka.

Kitas galimas problemos sprendimo būdas yra *iframe* žymų (angl. *tag*) naudojimas. Tačiau jas naudoti ne visada patogu ir įmanoma.

Trečias variantas, kurį siūlo RSH (angl. *Really Simple History*) biblioteka remiasi naršymo istorijos saugojimu *Map* tipo duomenų struktūroje, susiejant tam tikras sritis su atitinkamais duomenimis. Veiksmų ir naršymo istorijos pasikeitimų stebėjimui yra naudojama *hidden iframe* žyma. Pasikeitus tinklalapio turiniui pasikeičia ir naršyklės URL (angl. *Uniform Resource Locator*) adresas. Jam pridedamas identifikatorius, pvz. *#nauja-sritis.* Atitinkamai sistemos URL adresas pasikeičia iš

http://internetinė-sistema.org/ajax_pagrindu_veikianti_aplikacija

į naują adresą

http://internetinė-sistema.org/ajax_pagrindu_veikianti_aplikacija#nauja-sritis

Verta paminėti, kad toks sprendimas padeda išvengti dar vienos problemos – adresų saugojimo. Kadangi paprastai AJAX pagrindu veikiančiose internetinėse sistemose vienaip ar kitaip pakeitus tinklalapio turinį naršyklėje esantis URL adresas nepakinta, nėra galimybės išsaugoti nuorodą į einamąjį puslapį ir vėliau nuo ją atvėrus tęsti darbą arba tiesiog perduoti nuorodą kitiems asmenims. Tuo tarpu jeigu pasikeitus būsenai pakeičiamas ir URL adresas, tokia nuoroda gali būti išsaugoma ar kopijuojama. [Neu05]

### 3.1.2 Sprendimo realizacija

Esminis dalykas siekiant išspręsti naršyklės navigacijos mygtukų panaudojimą ir adresų saugojimą yra URL adreso ir internetinės sistemos būsenos abipusis susiejimas taip, kad pakeitus URL adresą atitinkamai pasikeistų ir sistemos būsena, o atsiradus būsenos pasikeitimams tai atsispindėtų ir URL adrese. Toks funkcionalumas turi būti realizuojamas panaudojant tam tikrą aibę savybių, kurių užtenka reikiamai būsenai aprašyti. Pavyzdžiui, jeigu sistemoje pateikiama elektroninė knygos kopija, tai konkretus vartotojui pateikiamas jos fragmentas gali būti apibūdinamas trimis savybėmis: skyriumi, skyriaus puslapiu bei kalba, kuria fragmentas pateikiamas. Užklausoje tai atsispindėtų taip:

www.knyga.lt#skyrius=3&puslapis=4&kalba=lietuviu

Sprendimui realizuoti reikia duomenų struktūros, kurioje būtų saugomos raktas-reikšmė poros, bei patogaus mechanizmo savybių reikšmėms susieti su URL adresu. Tam turi būti aprašytos tokios Javascript funkcijos:

1. Funkcija reikšmei iš duomenų struktūros gauti, kai žinomas raktas;
2. Funkcija rakto-reikšmės porai įtraukti į struktūrą;
3. Funkcija tikrinanti ar struktūroje jau yra pora su nurodytu raktu;
4. Funkcija užregistruoti objektams, kurių pasikeitimas turi inicijuoti atitinkamos reikšmės struktūroje atnaujinimą. Funkcija turi būti realizuojama panaudojant *attachEvent()* ir *addEventListener()* Javascript funkcijas (žiūrėti septintą priedą).
5. Funkcija užregistruoti objektams, kurie turi būti atnaujinami pasikeitus atitinkamoms reikšmėms struktūroje. Turi būti realizuota papildoma struktūra, kurioje būtų saugomos reikšmių poros savybė-objektų sąrašas. Tokiu atveju pasikeitus vienai iš savybių būtų galima inicijuoti su ja susietų objektų pasikeitimą.
6. Funkcija užregistruoti raktams, kurių reikšmės bus įtrauktos į URL adresą. Šiuo atveju reikės pagal funkcijų reikšmes suformuoti URL adreso eilutę, bei kraunant puslapį, ją išskaidyti į raktų-reikšmių poras.

Pirmosios trys sąrašo funkcijos atitinka įprastus *Map* struktūros metodus: *put(), get()* ir *hasKey()*. Duomenų struktūrai realizuoti turi būti naudojamas asociatyvus masyvas. *put()* ir *get()* funkcijos turi būti realizuojamos tiesiog priskiriant, gaunant reikšmes pagal raktą:

Struktūra[raktas] = reikšmė, arba

Reikšmė = struktūra[raktas].

Funkcija *hasKey()* realizuojama patikrinant ar pagal raktą gauta reikšmė nėra lygi *null*. [Her07]

## 3.2 Paieškos varikliukų problema

Ganėtinai aktuali AJAX pagrindu veikiančių sistemų problema yra tai, kad tokių tinklalapių nemato paieškos varikliai (pvz. Google). Tai nėra didelis trūkumas (o kartais netgi privalumas) sistemoms, kurios skirtos naudoti vietiniame tinkle arba jeigu ir yra prieinamos per internetą, tai tik siauram vartotojų ratui. Tačiau tai neretai sukelia problemų sistemų, kurios skirtos viešam naudojimui, kūrėjams ir vartotojams, nes paprastai norima, kad tokios sistemos būtų ne tik matomos paieškos varikliams, bet ir užimtų kuo aukštesnius reitingus.

Atsižvelgiant į paieškos variklių keliamus reikalavimus galima išskirti kelias veiksmų grupes: tokias kurias galima laisvai taikyti kuriamoje sistemoje nesibaiminant nesuderinamumo su paieškos varikliukais ir tokias, kurių viešam naudojimui skirtose sistemose geriau vengti.

Nekeliantys nesklandumų veiksmai:

- AJAX naudojimas specifiniams veiksmams vartotojo aplinkoje atlikti. Prie tokių veiksmų galima priskirti darbą su slapukais, sesijos apdorojimą bei žurnalizavimo veiksmus. Jeigu jie neįtakoja turinio, atitinkamai neturėtų kilti nesklandumų dėl nesuderinamumo su paieškos varikliukais;
- AJAX naudojimas turinio saugojimui. Formos, kurią vartotojas užpildė reikiamais duomenimis saugojimas taip pat neturi įtakos paieškos varikliukų darbui;
- AJAX naudojimas formos laukų validavimui. Formos laukų validavimas tiesiogiai neįtakoja tinklalapio turinio, taigi atitinkamai ir nesukelia minėtų problemų;
- AJAX naudojimas pranešimams apie atliktus veiksmus atvaizduoti. Pranešimuose vartotojui paprastai nebūna informacijos, kuri aktuali paieškos varikliams, todėl realizuojant šį funkcionalumą drąsiai gali būti naudojama AJAX technologija.

Veiksmai ir funkcionalumai, kuriems AJAX technologijos geriau nenaudoti, jeigu svarbus suderinamumas su paieškos sistemomis:

- Statinio teksto atvaizdavimas. Kalbant apie statinį tekstą turimas omenyje pagrindinis tinklalapio turinys, o ne kažkokie skaičiavimų rezultatai ar pan. Nes kaip tik tinklalapio turinys ir suteikia nemažą dalį informacijos paieškos sistemoms.
- Sąrašų bei lentelių puslapiavimas. Jeigu lentelėje ar sąraše yra pateikiamos kažkokių skaičiavimų reikšmės, AJAX technologijos naudojimas nėra aktualus, tačiau jeigu juose pateikiama tam tikra raktinė informacija apie sistemą, jie turėtų būti realizuoti tradiciniais būdais.
- AJAX technologijos naudojimas navigacijai. Verta paminėti, kad ši taisyklė gali būti taikoma ne tik AJAX, bet ir apskritai Javascript kalbą naudojančioms sistemoms. Kadangi paieškos varikliai navigacijai nenaudoja Javascript kalba paremtų nuorodų, bus pasiektas tik pagrindinis tinklalapis.

### 3.2.1 Galimi sprendimų variantai

Siekiant užtikrinti, kad internetinėje sistemoje esančios nuorodos bus pasiekiamos ir suprantamos paieškos varikliukams, jas reikia realizuoti panaudojant atributą *href* (<a href="..."/>). Tačiau skirtingi šaltiniai siūlo skirtingus realizacijos variantus. Vienas iš variantų realizuoti du atskirus tinklalapius: vieną suprantamą paieškos varikliukams, kitą skirtą naudojimui (pavyzdys: www.backbase.com). Tačiau akivaizdu, kad toks variantas yra ganėtinai neoptimalus ir nepatogus. Esant poreikiui tokią sistemą keisti praktiškai visada reiktų atlikti dvigubą darbą.

Kitas variantas – priklausomai nuo poreikių, toje pačioje sistemoje naudoti dviejų tipų nuorodas: tais atvejais, kai yra svarbu užtikrinti suderinamumą su paieškos varikliukais naudoti

*href* atributą, visais kitais – taikyti AJAX priemones. Pagrindinis šio sprendimo trūkumas yra tai, kad tam tikrais atvejais reikia užtikrinti, kad nuoroda bus suprantama paieškos varikliukams ir tuo pačiu ji bus realizuota panaudojant AJAX mechanizmą.

Tinkamiausias sprendimas būtų suderinti AJAX ir *href* atributo panaudojimą.

### 3.2.2 Sprendimo realizacija

Kadangi paieškos sistemos gali naudoti tik tradiciniu būdu aprašytas nuorodas, jos ir turi būti naudojamos tais atvejais, kai yra svarbu, kad sistema būtų pasiekiama paieškos varikliukams.

```
<a href="nuoroda1.html" title="Nuoroda" class="MenuElementas"> Nuoroda </a>
```

Tam, kad tinklalapio elementai būtų atnaujinami dinamiškai, navigacija turi būti atliekama panaudojant Javascript funkciją. Funkcijos priskyrimas turi būti vykdomas automatizuotai, panaudojant funkciją (žiūrėti aštuntą priedą).

Tokiu atveju jeigu turime tinklalapį, kurio adresas www.puslapis.lt paspaudus ant nuorodos, naujas URL adresas būtų http://www.puslapis.lt#http://www.puslapis.lt/nuoroda1.html. Verta paminėti, kad tokiu būdu realizavus navigaciją, adresus bus galima sėkmingai išsaugoti, taip pat veiks naršyklės navigavimo mygtukai [Tsc09].

## 3.3 Rezultatai

Su AJAX technologijos panaudojimu susijusios navigacijos problemos yra dvejopos: aktualios vartotojo interneto naršyklei ir aktualios internetinėms paieškos sistemoms. Pirmuoju atveju problema turi būti sprendžiama vienareikšmiškai susiejant internetinės sistemos būseną ir naršyklėje atvaizduojamą URL adresą. Kiekvieną kartą pasikeitus naršyklės būsenai, turėtų pasikeisti ir atvaizduojamas adresas. O kaskart nurodžius kitą URL adresą, turėtų pasikeisti ir internetinės sistemos būsena.

Paieškos varikliukų problema turi būti sprendžiama reikiamam elementui nurodant ir nuorodą (*href* atributą), ir jam priskiriant funkciją, kuri turi būti vykdoma tą elementą aktyvuojant.

# 4 Saugumo problemos

Praktiškai visos tradicinėms internetinėms sistemoms būdingos saugumo spragos yra aktualios ir AJAX pagrindu veikiančioms internetinėms sistemoms, o kai kurios iš jų pastarosioms gerokai pavojingesnės, kadangi tokio tipo sistemose gana daug operacijų atliekama kliento pusėje. Kai kurias iš saugumo problemų išsprendžią DWR karkasas, bet nemaža jų dalis lieka, todėl ganėtinai svarbi numatomo karkaso dalis yra skirta saugumo spragoms. Kadangi

papildomi funkcionalumai paprastai reiškia ir didesnį resursų poreikį, karkase turėtų būti paliekama galimybė sprendimų, skirtų saugumui užtikrinti, nenaudoti.

## 4.1 Įvedamų reikšmių validumo tikrinimas

Pasak internetinių sistemų saugumo ekspertų, būtų galima išvengti apie 80% internetinių sistemų atakų jeigu sistemose būtų tinkamai identifikuojami duomenys, gauti iš sistemos vartotojų. Nekorektiškų duomenų pateikimu remiasi *SQL Injection*, *Cross-site scripting* ir daugelis kitų atakų tipų.

### 4.1.1 Galimi sprendimų variantai

Klasikinis daugumos programuotojų sprendimas yra sudaryti sąrašą reikšmių, kurias aptikus duomenų eilutėje užklausa būtų identifikuojama kaip saugumo ataka ir sistema ją atmestų (angl. *blacklist*). Tačiau pramatyti visus galimus variantus yra ganėtinai sudėtinga ir dažniausiai neoptimalu.

Kur kas patogiau yra sudaryti reikalavimų sąrašą, kurias atitinkančios reikšmės būtų pripažintos kaip tinkamos naudoti (angl. *whitelist*) ir pagal tokius reikalavimus tikrinti reikšmių korektiškumą. Pvz., laukui, kuris skirtas elektroninio pašto adresui, gali būti keliami tokie reikalavimai:

- Vardo sritis turi būti sudaryta iš lotyniškų raidžių ir skaičių, joje gali būti taškų ir pabraukimo ženklų. Po kiekvieno taško ar pabraukimo ženklo turi būti raidė arba skaičius;
- Po vardo srities turi būti simbolis @;
- Po simbolio @ turi būti domeno sritis. Ši sritis turi būti sudaryta iš bent vieno teksto bloko, kurio turinį sudaro lotyniškos raidės ir skaičiai. Daugiausiai gali būti 3 teksto blokai, atskirti taškais ar pabraukimo ženklais;
- Turi būti nustatytas maksimalus adreso ilgis, taip išvengiant atminties perpildymo.

Taip pat siekiant supaprastinti tikrinimą, turi būti naudojama išraiškų kalba, pagal kurią aprašytas šablonas būtų lyginamas su gauta duomenų eilute. Siekiant užtikrinti, kad korektiškumo tikrinimas nebus neigiamai paveiktas sistemos vartotojo-kenkėjo, tikrinimas turi būti realizuotas serverio pusėje veikiančioje sistemos dalyje. [HofSul07]

### 4.1.2 Problemos sprendimas

Karkaso modelyje turi būti apibrėžtas mechanizmas leidžiantis patikrinti duomenų eilutės korektiškumą pagal duotą šabloną. Šablonų aprašymui turi būti naudojamos reguliariosios

išraiškos (angl. *regular expressions*). Saugumo sumetimais duomenų korektiškumo tikrinimas turi būti realizuotas serverio pusėje veikiančioje sistemos dalyje.

Kadangi reguliariųjų išraiškų naudojimas yra ganėtinai sudėtingas, turi būti pateikiami gatavi dažniausiai naudojami šablonai, tuo pačiu paliekant programuotojui galimybę aprašyti savo šablonus. Į paruoštų naudojimui šablonų sąrašą turi būti įtraukti:

- Tekstas. Simbolių eilutė, kurią sudaro tik raidės. Pvz., vardas, pavardė ar pan.;
- Skaičius. Simbolių eilutė, kurią sudaro tik skaitmenys. Eilutės pradžioje gali būti minuso ženklas;
- El. pašto adresas. Simbolių eilutė, kurioje yra @ simbolis (būtinai ne krašte). Po @ simbolio esančioje dalyje turi būti bent vienas taškas;
- Telefono numeris. Simbolių eilutė, kurią sudaro skaičiai ir tarpo simboliai. Eilutės pradžioje gali būti pliuso simbolis.
- Piniginė išraiška. Skaičių eilutė, kurios trečias nuo galo simbolis turi būti taškas arba kablelis (priklausomai nuo programuotojo nustatymų). Eilutės pradžioje minuso ženklo būti negali.
- Data ir/arba laikas. Simbolių eilutė atitinkanti dažniausiai naudojamus laiko ir/ar datos formatus. Pvz.: YYYY-MM-DD, hh:mm arba YYYY-MM-DD hh:mm:ss.

Serverio pusėje turi būti realizuojami atitinkami metodai (teksto korektiškumo tikrinimui, skaitinių reikšmių korektiškumo tikrinimui ir pan.). Tekstinei eilutei neatitinkant numatyto šablono turi būti grąžinamas klaidos pranešimas panaudojant išimtinių situacijų mechanizmą (angl. *exception handling*) [YosKeuSte+07].

## 4.2 Kodo slėpimas

Viena iš pagrindinių saugumo problemų AJAX pagrindu veikiančiose aplikacijose yra verslo logikos realizavimas Javascript kalba kliento pusėje. Kadangi Javascript kodas dažniausiai būna lengvai skaitomas, (o kartais dar ir su pakankamai išsamiais komentarais), paprastai vartotojams-kenkėjams nebūna labai sunku atlikti tam tikrus pakeitimus ir įvykdyti kodą pagal savo scenarijų. Kaip pavyzdį galima pateikti internetinės parduotuvės sistemą. Joje dalis verslo logikos yra realizuota Javascript kalba, veiksmai atliekami tokia tvarka:

- Patikrinti vartotojo autentiškumą;
- Gauti produkto kainą;
- Sužinoti vartotojo sąskaitos likutį;
- Pervesti pinigus iš vartotojo sąskaitos;
- Užsakyti produktą pristatymui.

Iš pirmo žvilgsnio toks algoritmas nekelia įtarimų. Visų pirma sistema patikrintų, prisijungusio vartotojo tapatybę, tada patikrintu ar vartotojas turi pakankamai pinigų ir jeigu taip, atskaičiavus produkto kainą, nurodytų pristatyti norimą produktą reikiamam vartotojui.

Tačiau verta prisiminti, kad didžioji dalis veiksmų atliekama kliento pusėje, vadinasi ir vartotojas panorėjęs gali:

- Tiesiog iškviesti funkciją, užsakančią produktą, apeinant vartotojo autentiškumo tikrinimą bei pinigų nuskaitymą;
- Atliekant pinigų pervedimo veiksmą, įvesti neteisingą sumą. Nors šiuo atveju logiškiau atrodytų tiesiog šio žingsnio išvengti, tačiau vartotojas gali įvesti neigiamą sumą. Tokiu atveju jeigu serverinėje sistemos dalyje nenumatytas apribojimas neigiamiems skaičiams, vartotojas gautų nemokamai norimą produktą, be to jo sąskaita būtų papildyta;
- Sužinoti bet kurio sistemos vartotojo sąskaitos likutį. Tai būtų pasiekiama funkcijai, kuri grąžina sąskaitos likutį pateikti kito vartotojo vardą.

### 4.2.1 Galimi sprendimų variantai

Kadangi Javascript kalba parašytas kodas yra vykdomas internetinėje naršyklėje, savaime suprantama, jis turi būti tai naršyklei prieinamas. O kodas prieinamas naršyklei, neišvengiamai bus prieinamas ir sistemos vartotojui. Dėl šios priežasties praktiškai nėra būdų, kaip padaryti kodą nepasiekiamą vartotojui. Tokiais atvejais siekiant apsaugoti Javasctipt kodą nuo neleistino koregavimo ar nuskaitymo jis turi būti šifruojamas. Užšifravus kodą jis tampa sunkiai skaitomas, tačiau išsaugo savo funkcionalumą.

Vienas iš siūlomų šifravimo variantų yra *unescape* funkcijos naudojimas ir teksto pateikimas vietoje simbolių nurodant jų kodus (tokiu būdu šifruojant, žodis *encoder* atrodytų taip: %45%6E%63%6F%64%65%72). Tačiau iš pažiūros saugus kodas labai nesunkiai gali būti iššifruojamas į skaitomą tekstą.

Pasižiūrėjus *Google maps* sistemos Javascript kodą pirmiausia į akis krenta tai, kad tekstas pateiktas be tarpų ar tabuliacijos ženklų, taip pat nėra kėlimo į naują eilutę. Funkcijų vardai sudaryti iš vieno simbolio, taigi yra visiškai neinformatyvūs. Taip parašytas tekstas yra gana sudėtingai skaitomas (ypač jeigu jis yra didesnės apimties) ir tik iš dalies gali būti konvertuojamas atgal, į patogiau skaitomą tekstą.

Siekiant didesnio saugumo, teksto šifravimas turi būti papildytas dar keliomis papildomomis funkcijomis.

### 4.2.2 Problemos sprendimas

Atitinkamai apdorojus Javascript funkcija virsta sunkiai skaitomu, iš pirmo žvilgsnio atsitiktinių simbolių kratiniu (žiūrėti devintą priedą).

Tokiam kodo transformavimui reikia atlikti šias operacijas:

- panaikinti kėlimo į kitą eilutę simbolius – \n;
- panaikinti komentarus;
- pašalinami nereikalingi tarpu ar tabuliacijos simboliai;
- kintamųjų ir funkcijų vardai pakeičiami, vardai prasidedantys didžiąja raide praleidžiami, darant prielaidą, kad tai yra klasių vardai, kurių keisti negalima;
- String tipo konstantose simbolius pakeisti \uXXX arba \XXX atitikmenimis;
- Skaičiai pakeičiami jų aštuntainiais atitikmenimis.

Svarbu paminėti, kad toks kodo šifravimas negarantuoja, kad fragmento perskaityti ir išanalizuoti bus nebeįmanoma ir saugumui jautrius funkcionalumus rekomenduojama realizuoti serverio pusėje veikiančioje sistemos dalyje.

## 4.3 Slapukų šifravimas

Slapukų (angl. *cookies*) naudojimas vienas iš dažniausiai klaidingai vertinamų aspektų interneto technologijose. Nemažai interneto vartotojų laikos klaidingos nuostatos, kad slapukų pagalba yra platinamos kenkėjiškos ar šnipinėjimui skirtos programos arba virusai. Tačiau iš tikrųjų patys internetinės sistemos vartotojai turi kur kas daugiau galimybių panaudoti slapukus blogiems tikslams.

Slapukuose neretai būna saugomi prisijungimui prie sistemos skirti duomenys bei sesijos kintamieji, be to slapukai saugomi vartotojo kompiuteryje ir dažniausiai nešifruojami. Siekiant užtikrinti, kad vartotojas nepakeis slapukuose saugomos informacijos savo naudai turi būti realizuotas mechanizmas slapukams užšifruoti.

Nepaisant kai kurių programuotojų nuomonės, kad slapukų šifravimas yra tik bereikalingas procesoriaus apkrovimas, yra nemažai priežasčių, kodėl turi būti pasirūpinta šia saugumo spraga:

1. Dauguma vartotojų kelioms sistemoms naudoja tą patį slaptažodį, todėl sužinojus vienos sistemos šifruotą slaptažodžio variantą, jis greičiausiai nesutaps su kitos sistemos šifruotu variantu. Taigi vartotojas-kenkėjas negalės apsimesti sistemos vartotoju, kurio slapuką jam pavyko gauti.
2. Neteisėtas slapukų kopijavimas gali būti vykdomas Javascript pagalba, išsiunčiant tam tikrą informaciją el. paštu, sistemos vartotojui to nežinant.

3. Atliekant kai kuriuos svarbesnius veiksmus (pvz. redaguojant vartotojo profilį ar keičiant slaptažodį) dauguma internetinių sistemų paprašo įvesti senąjį slaptažodį. Turint tik šifruotą jo versiją to padaryti negalima.

### 4.3.1 Galimi sprendimų variantai

Vienas iš variantų yra slapukuose šifruoti tik tam tikrus atskirus fragmentus, pvz. vartotojo slaptažodį. Tokiu atveju šifravimas (tada kai jo reikia) naudotų mažiau sistemos resursų, tačiau atitinkamai suteiktų kur kas mažiau saugumo.

Kitas variantas yra šifruoti visą slapuke talpinamą informaciją simetriniu raktu. Šis atvejis užtikrintų didesnį saugumą, bet kai kuriais atvejais vartotojui-kenkėjui gavus tokį slapuką juo būtų galima pasinaudoti.

Trečias variantas – į užšifruotą slapuką įtraukti vartotojo kompiuterio IP adresą ir kai kurią kitą informaciją (pvz. slapuko galiojimo laiką).

### 4.3.2 Problemos sprendimas

Karkaso modelyje turi būti apibrėžtas mechanizmas informacijai šifruoti. Pagal galimybes į slapuko turinį turi būti įtraukiama:

- Sesijos identifikatorius;
- Data ir laikas, kada slapukas buvo sukurtas;
- Galiojimo laikas. Apribojus galiojimo laiką, sumažėja grėsmė, kad neteisėtai nuskaitytas slapukas gali būti panaudotas;
- Vartotojo kompiuterio, kuriam sukurtas slapukas, IP adresas. Įtraukus IP adresą, slapukas gali būti panaudotas tik iš to kompiuterio, kuriame veikiančiai sistemai buvo sukurtas;
- Užkoduota informacija autentifikavimui (angl. *message authenticity check – MAC*).

MAC kodas reikalingas užtikrinti, kad nei vienas iš naudojamų laukų nebuvo pakeistas. Jam sudaryti turi būti naudojamas MD5 algoritmas. Saugumo mechanizmas turi veikti pagal schemą:

```
MAC = MD5("slaptas raktas " +
        MD5("sesijos identifikatorius" + "sukūrimo data" +
           "galiojimo laikas" + "IP adresas" +
           "slaptas raktas")
       )
```

Algoritmas sujungia visus slapuko duomenų laukus, tada prie jų prijungia slaptą raktą, žinomą tik internetinės sistemos serverinei daliai. Suformuota eilutė yra užkoduojama MD5 algoritmu. Prie gautos reikšmės dar kartą pridedamas slaptas raktas ir nauja eilutė užkoduojama

MD5 algoritmu pakartotinai. Pakartotinis kodavimas reikalingas tam, kad slapuką papildžius nauja informacija vartotojas-kenkėjas negalėtų duomenų perkoduoti.

Gauta reikšmė turi būti įtraukta į slapuko turinį. Vėliau, slapuko duomenis atsiuntus į serverinę sistemos dalį turi būti patikrinta ar nesibaigė galiojimo laikas ir ar IP adresas atitinka nurodytą. Tada anksčiau aprašytu būdu panaudojant MD5 turi būti sugeneruojamas kodas, kuris vėliau būtų palyginamas su saugomu slapuko turinyje. Jeigu jie sutampa – labai maža tikimybė, kad duomenų saugumas buvo pažeistas.

Pagal nutylėjimą visa informacija talpinama slapukuose turi būti šifruojama. [SteSte03]

## 4.4 Rezultatai

Verta paminėti, kad nemaža dalis saugumo problemų negali būti apeinama į karkaso sudėtį įtraukiant vieną ar kitą funkcionalumą. Didžioji dalis jų turi būti sprendžiama realizuojant serverio pusėje veikiančią sistemos dalį. Vis dėlto, kai kurie sprendimai gali ir turi būti įtraukti į karkaso sudėtį.

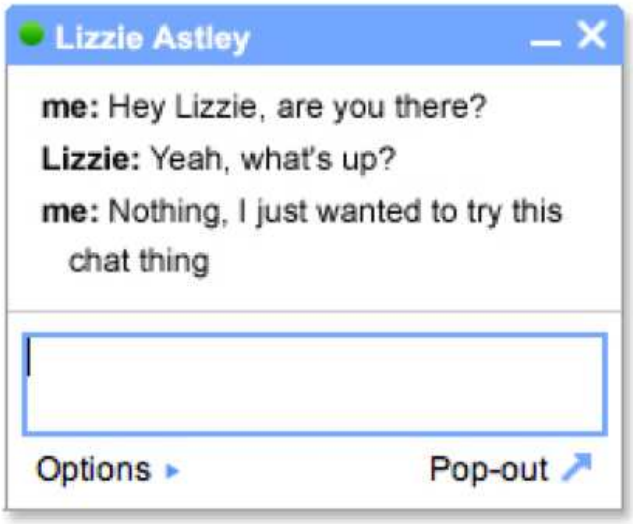
Reikšmių validumo tikrinimas turi būti atliekamas tikrinant ar vartotojo įvesta reikšmė atitinka iš anksto apibrėžtą šabloną. Šablonai nurodo, kokie simboliai gali būti pateiktose reikšmėse (pvz. tik skaitmenys).

Siekiant apriboti sistemos vartotojų galimybes matyti ir redaguoti vykdomą Javascript kodą, jis turi būti šifruojamas. Vis dėlto reikia paminėti, kad toks būdas tik iš dalies apsaugo sistemą. Ir saugumo atžvilgiu jautresnes sistemos dalis reiktų realizuoti serverio pusėje veikiančioje sistemos dalyje.

Kitas aprašytas funkcionalumas skirtas slapukų šifravimui. Jį panaudojus tampa ganėtinai sunku neteisėtai pasinaudoti slapukuose saugoma informacija.

# 5 Duomenų siuntimas iš serverio pusės

Internetinėse sistemose kliento ir serverio pusėse veikiančių sistemos dalių duomenų mainus inicijuoja kliento pusė. Iš čia siunčiama užklausa serverinei daliai, o serveriui suformavus ir grąžinus atsakymą jis apdorojamas ir atvaizduojamas. Tačiau kai kuriais atvejais iškyla poreikis funkcionalumo, kuriame duomenų mainus inicijuotų serverio pusė. Pvz., lentelė su tam tikrais duomenimis turėtų būti automatiškai atnaujinama jeigu kitas vartotojas sistemoje atliko tam tikrus pakeitimus arba internetinių pokalbių (angl. *chat*) lange, turėtų būti atvaizduojamas naujas tekstas, vos tik pašnekovas jį įveda.




*7 pav. Internetinio pokalbio langas*

Tiesiogiai inicijuoti duomenų mainų atlikimą serverio pusėje yra praktiškai neįmanoma. Taip yra todėl, kad HTTP protokolas leidžia tik būsenos neturinčią sąveiką, tarp sistemos dalių. Ji turi būti inicijuojama kliento pusėje, apdorojama serveryje ir vėl grąžinama klientui. Taigi HTTP protokolas tokio funkcionalumo nepalaiko.

### 5.1.1 Galimi sprendimų variantai

Tačiau yra įmanomas kitas problemos sprendimo variantas. Galima tam tikrais laiko tarpais inicijuoti kreipimąsi į serverio pusėje veikiančią sistemos dalį ir tikrinti ar nebuvo atliktas duomenų atnaujinimas ar kokie kiti aktualūs pakeitimai, o pastebėjus duomenų pakitimą juos atnaujinti kliento pusėje. Panašus sprendimas yra realizuotas *Echo2* karkase.

Egzistuoja keletas periodinių užklausų variantų: kai neaptikus pakeitimų serverinė sistemos dalis išsiunčia atitinkamą (galimai tuščią) pranešimą, arba kai neaptikus pakeitimų serverinė sistemos dalis užlaiko atsakymą. Verta pastebėti, kad esant dideliam sistemos vartotojų skaičiui atsakymo užlaikymas vartotų nemažai sistemos resursų.

### 5.1.2 Periodinių užklausų problema

Periodinių užklausų siuntimo mechanizmas turi būti ne tik efektyvus ir patikimas, bet ir naudoti kiek įmanoma mažiau sistemos aparatūrinės įrangos resursų. Vienas iš plačiausiai žinomų periodinių užklausų pavyzdžių yra elektroninio pašto programos periodinis kreipimasis į POP3 (Post Office Protocol) serverį. Paprastai naujos elektroninio pašto žinutės gaunamos kreipiantis į serverį praėjus tam tikram, iš anksto numatytam minučių skaičiui. Periodinių kreipimųsi problema yra tai, kad netinkamai nustačius kreipimosi laiką, svarbi informacija gali būti gauta ne laiku. Kiekvieną kartą neradus naujų laiškų, kliento pusei siunčiamas pranešimas „Naujų žinučių nėra“, tačiau jeigu naujas laiškas serverį pasiekia vos išsiuntus atitinkamą

pranešimą, sistemos vartotojas apie jį sužinos tik kitai užklausai grąžinus atsakymą. Tokiu atveju, jeigu užklausos siunčiamos kas dvi valandas, o praėjus 10 min. nuo užklausos serverį pasiekia laiškas, kuris vartotojui aktualus tik valandą, jis bus gautas pavėluotai.

Galimas tokios problemos sprendimas galėtų būti laiko intervalų tarp užklausų mažinimas, tačiau intensyvesnės užklausos neigiamai veiktų serverio ir kartu viso tinklo darbą. Tinklą atitinkamai paveiktų padidėjęs duomenų srautas, tuo tarpu serveriui tektų atsakyti į žymiai didesnį užklausų kiekį. Verta pastebėti, kad nustačius pernelyg mažą laiko intervalą didžioji dalis pranešimų būtų „Naujų žinučių nėra“.

Problema gali būti sprendžiama naudojant du duomenų mainams skirtus kanalus (angl. *stream*): duomenų gavimui ir duomenų siuntimui. Vienas iš kanalų galėtų būti naudojamas periodinių pranešimų siuntimui. Jeigu serveryje naujų pranešimų nėra – serveris į užklausą galėtų atsakyti ne iš karto. Tuo tarpu kitas kanalas galėtų būti naudojamas įprastiems duomenų mainams.

Yra trys pagrindiniai tokio pobūdžio duomenų mainų tipai:

#### 5.1.2.1 Būsenos atnaujinimas

Būsenos atnaujinimo (angl. *status updates*) pagrindas yra tam tikra klientinei sistemos daliai aktuali globali informacija, kuri laikoma serverinėje sistemos dalyje. Skirtingi sistemos vartotojai minėtus duomenis mato vienodai. Kaip pavyzdį galima pateikti akcijų vertės kitimų kreivę. Tokia informacija nėra skirta kažkokiam konkrečiam sistemos vartotojui, antra vertus visi sistemos vartotojai ją mato taip pat. Tokiam informacijos pakeitimui nereikia tapatybės nustatymo. Tai nereiškia, kad duomenys bus prieinami bet kam. Informacija prieinama grupei žmonių turinčių teisę ją matyti, tačiau sistemai nėra skirtumo kaip kokiam vartotojui tie duomenys turi būti pateikiami.

#### 5.1.2.2 Būsenos stebėjimas

Būsenos stebėjimas (angl. *presence detection*), kaip ir ankstesnis tipas, pasireiškia, kai vartotojų grupei yra aktualus vienas ir tas pats objektas. Atitinkamai tas objektas visiems pateikiamas vienodai. Skirtumas yra tas, kad objekto būsena priklauso nuo vartotojų. Kaip pavyzdį galima pateikti internetiniuose forumuose atvaizduojamą prisijungusių diskusijos dalyvių sąrašą. Visi dalyviai mato tuos pačius vardus, tačiau prisijungus naujam vartotojui arba atsijungus diskusijos dalyviui, sąrašas automatiškai keičiasi.

#### 5.1.2.3 Serverio duomenų siuntimas

Serverio duomenų siuntimas (angl. *server push*). Šiuo atveju nors ir kreipdamiesi į vieną ir tą patį duomenų objektą, vartotojai gauna skirtingai suformuotus rezultatus. Priklausomai nuo užklausos. Pavyzdžiui, sistemoje pateikiančioje informaciją apie akcijų kainas, nepriklausomai nuo to, kad duomenys visiems sistemos vartotojams yra vienodi, kiekvienas vartotojas gali užsisakyti ir gauti tik jį dominančių akcijų kainas. Atitinkamai, šiuo atveju skirtingai nuo ankstesnių dviejų, būtina nustatyti kliento tapatybę.

Nepaisant to, kad trečiasis tipo pavadinimas – serverio duomenų siuntimas, praktiškai visi trys tipai realizuoja serverio duomenų siuntimą. Yra du požymiai pagal kuriuos galima skirti tipus: kaip yra atvaizduojami duomenys ir ar reikia nustatyti sistemos vartotojo tapatybę.

### 5.1.3 Būsenos atnaujinimas

Būsenos atnaujinimai paprastai būna globalūs. Tai jokiu būdu nereiškia globalių duomenų ar globalių kintamųjų. Turima omenyje tai, kad duomenys, pasiekiami šiuo būdu yra pasiekiami visiems. Tokia informacija yra vienoda visiems sistemos vartotojams.

Sistemos vartotojų atžvilgiu tokie duomenys laikomi labiau skaitymui skirtais duomenimis (angl. *read-mostly data*). Labiau skaitymui skirti duomenys tik išimtiniais atvejais yra atnaujinami ar keičiami sistemos vartotojų. Dažniausiai jų pakitimus inicijuoja išoriniai faktoriai.

#### 5.1.3.1 Klientinė sistemos dalis

Klientinėje sistemos dalyje turi būti apibrėžtas *ClientComunicator* objektas. Šios dalies funkcijos yra apibrėžti URL adresą, aprašyti grįžtamojo kvietimo funkciją bei realizuoti periodinį kreipimąsi į serverį (žiūrėti dešimtą priedą).

Pateikto pavyzdžio pradžioje sukuriamas *ClientCommunicator* objektas ir priskiriamas kintamajam *client.* URL adresas, kuriuo bus siunčiami ir gaunami duomenys nurodomas objekto parametrui *client.baseURL*. Parametras *client.listen* gauna reikšmę funkcijos, kuri, panaudojant *ClientCommunicator* objektą, kviečiama kiekvieną kartą, kai serverinė sistemos dalis atsiunčia naujus duomenis.

StartCommunications() funkcija inicijuoja duomenų mainų pradžią, EndCommunications() stabdo periodinį kreipimąsi į serverį, o SendData() funkcija inicijuoja duomenų siuntimą iš klientinės sistemos dalies į serverinę dalį. Metodas *client.send()* atlieka duomenų siuntimą į serverį. Pirmuoju parametru jame aprašomas siunčiamo turinio MIME (angl. *Multipurpose Internet Mail Extensions*) tipas, antrasis – siunčiamų duomenų ilgį, o trečiasis – pačius duomenis.

### 5.1.3.2 ClientCommunicator objektas

*ClientCommunicator* objektas turi turėti šiuos parametrus:

- *baseURL*, kuriame būtų saugomas URL adresas duomenų mainams;
- *username*, kuriame būtų saugomas sistemos vartotojo vardas;
- *password*, kuriame būtų saugomas sistemos vartotojo slaptažodis;
- *listen*, kuriame būtų saugoma duomenų mainams skirta funkcija;
- *doLoop*, kuriame būtų nurodoma ar turi būti tęsiamas periodinis duomenų tikrinimas;
- *callDelay*, kuriame būtų nurodoma, kokios trukmės delsa turi būti po kiekvieno gauto atsakymo prieš pakartotinį užklausos kvietimą;
- *instances*;
- *index;*
- *instanceCounter*.

*CallDelay*, *instances*, *index* ir *instaceCounter* kintamieji naudojami anksčiau minėtai periodinių užklausų problemai spręsti. Sprendimui negali būti taikomas įprastas Javascript ciklo mechanizmas, nes išsiuntus užklausą, sistemos darbas sustotų, kol nebus gautas atsakymas. Naudojant asinchronines užklausas, darbas nesustotų, tačiau tokių užklausų būtų išsiunčiama be galo daug. Problemos priežastis yra tai, kad Javascript nepalaiko lygiagretaus kelių procesų vykdymo. Kelių funkcijų išlygiagretinimas gali būti imituojamas atliekant veiksmų seką:

- Iškvietus reikiamą funkciją, vykdomas kreipimasis į serverį, tam panaudojant duomenų gavimui skirtą kanalą;
- Įvykdžius kreipimąsi į serverį funkcija baigia darbą. Naršyklė gali toliau sėkmingai siųsti ir priimti duomenis;
- Serverinėje sistemos dalyje tuo metu formuojamas atsakymas į užklausą, jeigu reikia, atsakymas užlaikomas;
- Grąžinus rezultatą į serverinę sistemos dalį, atsakymas apdorojamas ir po nurodyto laiko intervalo kartojama užklausa kepimuisi į serverį.

Kviečiant funkciją nurodytais laiko intervalais, naudojamas *window.setTimeout* metodas, kuriam kviečiama funkcija turi būti perduodama kaip tekstinė eilutė.

Tačiau toks sprendimas ne visada geras. Kintamasis *value* yra nuoroda į objektą ir atliekant objektų serializavimą sukuriami keli objektai, kurie tarpusavyje gali skirtis. Kur kas patogiau yra perduoti masyvo, į kurį talpinamos objektų reikšmės, indeksą. Tokiam kintamųjų saugojimui ir yra naudojami kintamieji *callDelay*, *instances*, *index* ir *instaceCounter* (žiūrėti vienuoliktą priedą).

Kintamasis *instances* yra masyvas, susietas su *prototype* objekto kintamuoju. Tai reiškia,

kad kiekvieną kartą kuriant *ClientCommunicator* objektą bus naudojamas vienas ir tas pats masyvas. Kintamasis *index* nurodo, kuris *ClientCommunicator* objektas yra naudojamas, o *counter*, kiek tokių objektų yra iš viso.

Siunčiant periodines užklausas dažniausiai yra aktuali ne visa serveryje esanti informacija, o informacija pakitusi po paskutinio siuntimo. Tokiu atveju serverinėje sistemos dalyje turi būti realizuotas skaitliukas (*long* tipo kintamasis), kurio reikšmė padidėtų kiekvieną kartą atlikus pakeitimus. Tuo tarpu kliento pusėje turėtų būti atitinkamas kintamasis su naujausia turima versija. Minėtas kintamasis turėtų būti tikrinamas kiekvieną kartą siunčiant užklausą serveriui ir pastebėjus, kad serverio pusėje esančio kintamojo reikšmė yra didesnė turi būti inicijuojamas duomenų atnaujinimas. [Gro06]

## 5.2 Rezultatai

Kadangi HTTP protokolas leidžia realizuoti tik tradicinį duomenų mainų modelį, t.y. kai užklausa inicijuojama klientinės sistemos dalies. Priešingą atvejį, kai iniciatyva priklauso serverinei sistemos daliai tenka imituoti. Tokiu atveju klientinė sistemos dalis periodiškai kreipiasi į serverinę dalį ir tikrina ar nebuvo atlikti pakitimai.

# 6 Klaidų apdorojimas

Klaidų apdorojimo mechanizmas turi užtikrinti, kad sistemoje įvykus klaidai ji būtų laiku identifikuota, bei apie ją būtų pranešama. Verta paminėti, kad sistemos vartotojams ir ją prižiūrintiems asmenims turi būti siunčiami skirtingi pranešimai. Priešingu atveju prižiūrintiems asmenims gali būti sunku identifikuoti klaidos priežastis, o sistemos vartotojams gali būti pateikta per daug informacijos apie sistemos sandarą, kas atitinkamai keltų grėsmę saugumui.

## 6.1 Užklausos trukmės ribojimas

Ne visada pavyksta sėkmingai įvykdyti užklausą bei priimti jos rezultatą. Tai gali įvykti dėl tinklo delsos, dėl sutrikusio serverinės dalies veikimo ar dėl kitų problemų. Tokie atvejai, nors ir pasitaikantys ganėtinai retai, taip pat turi būti numatyti bei atitinkamai turi būti aprašyti veiksmai, kurie bus atlikti įvykus tokiam nesklandumui.

Svarbiausias dalykas yra duomenų perdavimo sutrikimo identifikavimas. Vienas iš būdų identifikuoti tokio tipo problemas yra užklausos trukmės ribojimas. Negavus atsakymo per numatytą laiko tarpą (pvz. 10 sekundžių) būtų laikoma, kad užklausa nebuvo sėkmingai įvykdyta. Tokiu atveju būtų kviečiama atitinkama Javascript funkcija, kurioje būtų aprašyti alternatyvūs veiksmai (išvedamas pranešimas apie klaidą, pakartotinis užklausos kvietimas ir pan.).

### 6.1.1 Problemos sprendimas

Kiekvieną kartą išsiuntus užklausą servinei sistemos daliai, turi būti iškviečiama alternatyvių veiksmų užklausa, kaip parametrą jei perduodant laiką milisekundėmis, kada turi būti atliekami joje aprašyti veiksmai (jeigu laiko trukmė nurodyta neigiama, alternatyvūs veiksmai nebūtų kviečiami). Papildomai turi būti aprašytas *boolean* tipo kintamasis, kurio reikšmė grįžus užklausos atsakymui būtų pakeista iš *false* į *true*. Alternatyvių veiksmų funkcijos veikimas ją iškvietus būtų sustabdomas numatytam laiko tarpui, o tada tikrinama minėto kintamojo reikšmė. Jeigu reikšmė būtų lygi *false*, turėtų būti vykdomi numatyti alternatyvūs veiksmai.

## 6.2 Nenumatytų situacijų apdorojimas

Internetinėje sistemoje įvykus klaidai arba nenumatytai situacijai (angl. *exception*) svarbu užtikrinti ne tik tolesnį sistemos funkcionavimą, bet ir numatyti, kad išvestas klaidos pranešimas neatskleistų pernelyg daug informacijos apie sistemos veikimo principus. Tai ypač aktualu viešai prieinamoms sistemoms, nes vartotojai-kenkėjai gautą informaciją gali panaudoti blogiems tikslams.

Kurdami ir testuodami sistemą, programuotojai paprastai naudoja kiek įmanoma išsamesnius klaidų pranešimus. Tai leidžia greičiau aptikti klaidas ir trūkumus, nes klaidų pranešimai paprastai pateikia kurioje tiksliai vietoje ir kokiomis aplinkybėmis sutriko sistemos darbas.

Problemų kyla tada, kai programuotojai palieka pernelyg išsamius klaidų pranešimus produkcinėje sistemos versijoje. Dažniausiai taip daroma siekiant sudaryti galimybes greičiau ir efektyviau atsižvelgti į kilusias problemas. Tokiu atveju sistemai išvedus klaidos pranešimą, programuotojas neretai iškart gali identifikuoti trūkumą.

### 6.2.1 Galimi sprendimų variantai

Informacijos apie klaidas pateikimo variantai gali būti skirstomi pagal apimtį (išsamūs ar glausti) ir pagal pateikimo būdą (žurnalizavimas, pateikimas į vartotojo sąsają, išsiuntimas el. paštu). Gali būti taikomos įvairios kombinacijos.

### 6.2.2 Problemos sprendimas

Siekiant išvengti minėtos problemos, karkasas turi realizuoti klaidų apdorojimo mechanizmą veikiantį dviem rėžimais: sistemos kūrimo ir sistemos naudojimo. Pirmuoju atveju būtų pateikiami išsamūs klaidų pranešimai, leidžiantys nustatyti klaidos vietą ir neturintys saugumo apribojimų. Antruoju atveju klaidos pranešimai turėtų būti lakoniški, trumpai

apibūdinantys klaidą ir nenurodantys kokioje sistemos dalyje ir kokiomis aplinkybėmis ji įvyko. Galimi klaidų pranešimų naudojimo scenarijai:

- Priklausomai nuo sistemos parametro yra įjungiamas vienas arba kitas klaidų apdorojimo atvejis. Svarbu užtikrinti, kad parametras galėtų būti keičiamas tik serverio pusėje ir sistemos vartotojams nebūtų prieinamas;
- Vartotojui išvedamas lakoniškas klaidos pranešimas, tuo tarpu išsamus klaidos aprašymas išsaugomas sistemos žurnale (angl. *log*) arba išsiunčiamas nurodytu el. paštu;
- Vartotojui pateikiamas klaidos pranešimas nurodantis klaidos kodą arba numerį. Kodų atitikmenys būtų pateikiami atskirame dokumente. [Had06]

Karkaso modelyje turi būti numatyti keturi klaidų išvedimo variantai:

- Išsamaus pranešimo išvedimas vartotojo sąsajoje;
- Glausto pranešimo išvedimas vartotojo sąsajoje;
- Elektroninio laiško siuntimas;
- Klaidos pranešimo saugojimas sistemos žurnale.

Kai kurie iš jų gali būti apjungiami. Pranešimo tipą galima apibrėžti trimis loginio tipo reikšmėmis, kurias galima apjungti į masyvą. Pirmas elementas nurodytų ar vartotojo sąsajoje pateikiamas išsamus pranešimas (priešingu atveju būtų pateikiamas glaustas pranešimas), antrasis – ar reikia siųsti elektroninį laišką, trečiasis – ar pranešimas turi būti traukiamas į sistemos žurnalą.

## 6.3 Rezultatai

Klaidų apdorojimo mechanizmas apima du atvejus: kai serverinė dalis negrąžina jokio rezultato arba kai serverio pusėje veikianti sistemos dalis grąžina klaidos pranešimą.

Pirmuoju atveju problema sprendžiama nustatant laiką, kurį bus laukiama atsakymo į užklausą. Jei per nurodytą laiką negrąžinamas atsakymas, turi būti vykdomi alternatyvūs veiksmai.

Antruoju atveju apibrėžiami keli klaidų išvedimo variantai, kurie gali būti taikomi priklausomai nuo to ar sistema yra kūrimo, testavimo ar produkcinėje fazėje.

# 7 Būsenos apdorojimas

AJAX technologiją naudojančiose internetinėse sistemose dėl dalinio tinklalapių atnaujinimo paprastai neveikia tradicinėms sistemoms būdingas mechanizmas, kai naršyklė kaupia aplankytus tinklalapius ir prireikus juos užkrauna panaudodama išsaugotus resursus.

Kita problema kyla dėl to, kad dalis sistemos būsenos saugoma kliento pusėje veikiančioje sistemos dalyje, o duomenys tarp kliento ir serverio pusėse veikiančių sistemos dalių ne visada tinkamai sinchronizuojami.

## 7.1 Podėliavimas

Paprastai naršyklės vienaip ar kitaip podėliuoja (angl. *caching*) užkrautus tinklalapius sistemos vartotojo kompiuteryje. Todėl atsiradus poreikiui pakartotinai užkrauti tinklalapį, kuris neseniai buvo užkrautas, sistema dažniausiai tiesiog nuskaito reikiamą tinklalapį iš podėlio (angl. *cache*), taip sutaupydama laiko ir resursų.

Kalbant apie AJAX pagrindu veikiančias aplikacijas verta paminėti, kad čia dažniausiai pakeičiamas tik tam tikras tinklalapio fragmentas. Naršyklė negali tokio pakeitimo užfiksuoti ir atitinkamai į podėlį nėra įtraukiamas atitinkamas įrašas. Norint sutaupyti laiko ir resursų, reikia realizuoti podėlį internetinėje sistemoje.

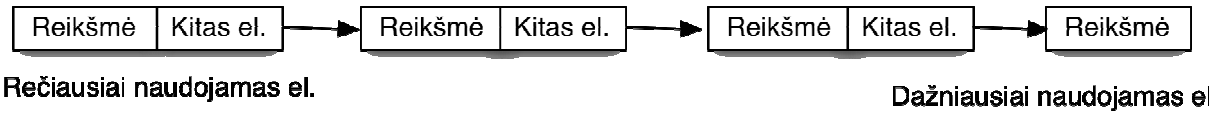
### 7.1.1 Galimi sprendimų variantai

Pagrindiniai reikalavimai podėliui yra efektyvus naujo įrašo saugojimas bei greita ir paprasta reikiamo įrašo paieška. Vienas iš geriausių siūlomų variantų duomenims saugoti yra *Map* tipo duomenų struktūra arba jos atitikmuo Javascript kalboje – asociatyvus masyvas. Kviečiant XmlHttpRequest tipo užklausą paprastai yra naudojami dviejų tipų duomenys: URL adresas ir parametrai perduodami tam tikram Java metodui. Juos apjungus gaunama reikšmė, kurią galima naudoti kaip masyvo raktą. Tokiu atveju prieš kviečiant užklausą, reiktų patikrinti ar masyve nėra įrašo, kurio raktas atitiktų suformuotą pagal URL adresą ir parametrus ir jeigu toks įrašas yra, užuot kvietus užklausą, naudoti įrašo reikšmę.

Tačiau norint sprendimą taikyti didesnės apimties internetinėse sistemose reikalingi tam tikri jo pataisymai, nes Javascript masyvas greitai persipildytų, o tai gali turėti neigiamos įtakos sistemai. Problemos sprendimas būtų apriboti masyvo dydį, o masyvui pasiekus ribą, šalinti anksčiausiai įtrauktus įrašus. Anksčiausiai įtrauktų įrašų identifikavimas turėtų būti atliekamas taikant mažiausiai naudojamų įrašų (angl. *Least Recently Used*) algoritmą. Tokiu atvejų būtų sudaromas sutvarkytas sąrašas, kurio pradžioje būtų seniausiai įtrauktas (arba rečiausiai naudojamas) įrašas, o pabaigoje – vėliausiai įtrauktas (arba dažniausiai naudojamas) įrašas. Žinoma, kai kuriais atvejais yra paranku turėti galimybę keisti masyvo dydį, taigi į tai taip pat turėtų būti atsižvelgta. [Per06]

### 7.1.2 Problemos sprendimas

Klientinėje dalyje turi būti realizuota sutvarkyto sąrašo tipo struktūra, atitinkanti iliustraciją.




*8 pav. Sutvarkyto sąrašo struktūra*

Struktūra turi turėti metodą saugomoms reikšmėms gauti pagal raktą, kurį sudaro URL adresas ir naudojami parametrai ir metodą saugomoms reikšmėms įtraukti į struktūrą. Kiekvieną kartą kreipiantis į serverinę sistemos dalį turi būti kviečiamas pirmasis metodas. Jo turinys:

```
if(structure[URL + parameters]) {
  response = structure[URL + parameters];
}
else{
  //Podėlyje reikiamos reikšmės nėra, kreipiamės į serverį
...
}
```

Atitinkamai, kiekvieną kartą gavus atsakymą į užklausą, jis turi būti išsaugomas:

```
structure[URL + parameters] = httpreq.responseText;
```

## 7.2 Būsenos saugojimas

Netradicinis veikimo principas AJAX pagrindu veikiančiose sistemose turėjo įtakos informacinės sistemos būsenos (angl. *state*) saugojimui. Kadangi nemaža dalis sistemos atliekamų funkcijų yra perkelta į klientinę dalį, svari duomenų dalis būna saugoma tiesiog Javascript kintamuosiuose. Tai reiškia, kad perkrovus tinklalapį duomenys dingtų, jeigu prieš tai nebūtų išsaugoti serverio pusėje.

Egzistuoja keli variantai duomenims kliento pusėje saugoti. Vienas iš tokių variantų – slapukų naudojimas. Tačiau juose ne galima saugoti tik tam tikrą informaciją. Pavyzdžiui, nerekomenduojama saugoti slaptos informacijos arba itin didelių duomenų kiekių, nes tai paprastai turi neigiamos įtakos tinklo darbui.

Dažniausiai visa reikiama informacija saugoma sesijos kintamuosiuose serverio pusėje, o kliento pusėje esančiame slapuke pateikiamas tik sesijos identifikatorius, leidžiantis konkrečiam sistemos vartotojus priskirti tik jam aktualius duomenis. Sesijų naudojimas yra įprastas dalykas tradicinėse internetinėse sistemose, tačiau jas naudojant AJAX pagrindu veikiančiose programose reikia užtikrinti efektyvius duomenų mainus tarp kliento ir serverio pusių.

### 7.2.1 Periodinis serveryje esančių duomenų tikrinimas

Ši problema aktuali internetinėms sistemoms, kuriose realizuoti akcijų kursų atnaujinimas, sporto varžybų rezultatai ar kita nuolat besikeičianti informacija. Vienas iš galimų problemos sprendimų – duomenų rašymas į atskirą failą. Tai yra ganėtinai nesudėtingai realizuojama, tačiau gali būti taikoma tik tokiais atvejais, kai kliento pusei reikalingi duomenys nėra susiję su informacija, kuri yra būdinga tik konkrečiam vartotojui. Kitas atvejis sudėtingesnis, jam reikia realizuoti mechanizmą sekantį vartotojo veiksmus, ir juos registruojant įtraukti laiko įrašą (angl. *timestamp*). Kaskart atnaujinant duomenis reiktų tikrinti laiko įrašą su esama data.

### 7.2.2 Galiojimo laikas

AJAX pagrindu veikiančiose sistemose pasitaiko nesklandumų dėl pasibaigusio sesijos galiojimo laiko. Paprastai tokiais atvejais lieka nutrūksta ryšys tarp klientinės ir serverinės dalies, duomenys lieka neišsaugoti, vartotojas atjungiamas nuo sistemos, tačiau neužkraunamas prisijungimo puslapis, taigi sistemos vartotojas pats turi inicijuoti jo užkrovimą. Problemai spręsti turi būti realizuoti du papildomi funkcionalumai:

- Iki sesijos galiojimo pabaigos laikus tam tikram iš anksto numatytam laiko tarpui, vartotojas turi būti apie tai informuojamas, jeigu įmanoma turi būti pasiūloma išsaugoti tinklalapio duomenis;
- Turi būti realizuotas prisijungimo prie sistemos puslapio užkrovimas, kuris būtų vykdomas pasibaigus sesijos galiojimo laikui, kad iš naujo prie sistemos prisijungęs vartotojas galėtų sėkmingai tęsti darbą.

### 7.2.3 Kitos su sesijų naudojimu susijusios problemos

- Duomenų praradimas. Informacija gali būti prarandama užlūžus vartotojo naršyklei, sutrikus serverio darbui arba dėl nepatikimo duomenų perdavimo kanalo funkcionavimo sutrikimo;
- Kada reikia atlikti duomenų mainus? Galimi du atvejai: siųsti duomenis į serverio pusę kaskart kai tik vartotojas atlieka pakeitimus arba siųsti visą duomenų paketą (pvz. paspaudus mygtuką „Saugoti“);
- Javascript vykdymo aplinka. Paprastai programuotojai nenumato galimo Javascript šiukšlių surinkėjo (angl. *garbage collector*) įsikišimo. Dėl šio funkcionalumo nerekomenduojama klientinėje sistemos dalyje saugoti didelių duomenų kiekių, nes automatiškai suveikus šiukšlių surinkėjui, jie gali būti prarasti;
- Vartotojų konkuravimas. Problema kyla kai du ar daugiau sistemos vartotojų atlieka keitimus viename ir tame pačiame objekte. Tokiu atveju antrasis vartotojas gali

perrašyti pirmojo atliktus pakeitimus. Vienas iš galimų problemos sprendimų galėtų būti redaguojamo įrašo „rakinimas“ taip padarant jį prieinamą tik vienam iš sistemos vartotojų.

### 7.2.4 Problemos sprendimas

Karkaso modelyje turi būti numatyta automatinio saugojimo galimybė, kuri periodiškai vykdytų kliento pusės būsenos saugojimą serverio pusėje. Serverinėje sistemos dalyje turėtų būti realizuotas buferis, į kurį būtų talpinamas objektas, atspindintis kliento pusės būseną. Konkrečiai tai turėtų būti DOM objekto fragmentas, kurio identifikatorius būtų iš anksto apibrėžtas.

Kliento pusėje turėtų būti skaičiuojamas laikas, nuo paskutinio sistemos vartotojo atlikto veiksmo ir likus tam tikram periodui iki sesijos pabaigos, turi būti išvedamas pranešimas vartotojui. Papildomo saugojimo realizuoti nereikia, nes duomenys ir taip būtų talpinami į buferį serverio pusėje praėjus tam tikram laiko etapui.

## 7.3 Rezultatai

Podėliavimo problema turi būti sprendžiama realizuojant riboto dydžio duomenų struktūrą, kurioje būtų talpinami tinklalapio būseną atspindintys duomenys. Kiekvieną kartą prieš kraunant tinklalapį ar reikiamų duomenų nėra aprašytoje struktūroje.

Būsenos sinchronizavimo problema turi būti sprendžiama periodiškai išsaugant klientinės sistemos dalies būseną serverio pusėje. Tokiu atveju būtų realizuota galimybė nuo informacijos praradimo sutrikus ryšiui tarp sistemos dalių.

# 8 Išvados

Dauguma darbe minimų problemų, kurios iškyla taikant AJAX technologiją yra jau ganėtinai seniai žinomos. Kai kurios iš jų yra lygiai taip pat aktualios ir tradicinėms, AJAX technologijos nenaudojančioms, internetinėms sistemoms. Atitinkamai, nemaža jų dalis jau turi vienokius ar kitokius sprendimus. Tačiau tokie sprendimai neretai būna pritaikyti tam tikroms konkrečioms situacijoms ir ne visada yra tinkami naudoti. Juo labiau jie negalėtų būti taikomi kaip bendro pobūdžio sprendimai. Dėl šios priežasties norint gauti bendram naudojimui tinkamą technologiją sprendimus reikia modifikuoti arba tiesiog apjungti.

DWR technologija puikiai tinka duomenų mainams atlikti, tačiau ji neturi beveik jokių papildomų funkcijų. Antra vertus ją panaudojus kaip pagrindą ir suteikus reikiamus funkcionalumus gaunamas modelis apimantis didžiąją dalį sprendimų, reikalingų pagrindiniams AJAX technologijos nesklandumams apeiti. Toks modelis apima tiek patį duomenų mainų mechanizmą, tiek sudėtingesnius ir labiau aktualius vartotojo sąsajos aspektus.

Viena iš ryškesnių AJAX technologiją naudojančių sistemų problemų – vartotojo sąsajos realizavimas sprendžiamas apjungiant kelis sprendimus: paprastas bazinių elementų susiejimas, pateikiami sudėtingesni komponentai bei paliekama galimybė programuotojui realizuoti vartotojo sąsają savarankiškai.

Karkaso modelis apima ir ganėtinai plačiai žinomų navigacijos problemų sprendimus. Tiek naršyklės mygtukų veikimą, tiek suderinamumą su paieškos varikliukais.

Verta paminėti, kad kai kurių problemų sprendimai yra bendro pobūdžio ir jie gali būti naudojami tik kaip patarimai programuotojams. Jie negali būti suvesti į vieną bendram naudojimui tinkančią karkaso dalį, ar kažkokį konkretų algoritmą. Tai ypač aktualu kalbant apie sprendimus sistemos saugumui užtikrinti.

Vis dėlto didžioji dalis problemų, kylančių dėl AJAX technologijos naudojimo gali būti sėkmingai sprendžiamos taikant atitinkamus algoritmus arba atliekant reikiamas operacijas.

# 9 Literatūros apžvalga

[Alm05] Dion Almaer. Utility: JavaScript Obfuscator. http://ajaxian.com/archives/utility-javascript-obfuscator. 2005. 66.31 KB

[Ber07] Anthony Bernard. Google Web Toolkit - Build AJAX apps in the Java language. http://code.google.com/webtoolkit/. 252 KB

[Bra05] Joe Bradbury. DWR: Easy AJAX for JAVA. http://getahead.ltd.uk/dwr. 293 KB

[Car05] Cloves Carneiro Jr. AJAX made simple with DWR. http://www.javaworld.com/javaworld/jw-06-2005/jw-0620-dwr.html. 352 KB

[Cla06] Michael Clark. Building Next Generation Web Applications using JSON-RPC-Java. http://oss.metaparadigm.com/jsonrpc-dist/json-rpc-java-talk-20050225.pdf. 331 KB

[Gro06] Christian Gross. Ajax Patterns and Best Practices. Apress. 2006. 409 psl.

[Gut09] Scott Guthrie. jQuery Plugin for Requesting Ajax-like File Downloads. http://www.filamentgroup.com/lab/jquery_plugin_for_requesting_ajax_like_file_downloads/. 2009. 7.54 KB

[Had06] Kris Hadlock Ajax for Web Application Developers. Sams. 2006. 288 psl.

[Her07] Matthias Hertel. Aspects of AJAX. Publikacija internete. 2007. 142 psl.

[HofSul07] Billy Hoffman, Bryan Sullivan. Ajax security. Addison Wesley. 2007. 504 psl.

[Lie06] Tod Liebeck. Comparing the Google Web Toolkit to Echo2. http://echotwo.blogspot.com/2006/05/comparing-google-web-toolkit-to-echo2.html. 219 KB

[McC05] Philip McCarthy. Ajax for Java developers: Ajax with Direct Web Remoting. http://www-128.ibm.com/developerworks/java/library/j-ajax3. 421 KB

[Mer05] John Meriald. Introducing JSON. http://www.json.org. 143 KB

[Mer06] Mario Merlusconi. Echo2. http://www.nextapp.com/platform/echo2/echo/. 134 KB

[Neu05] Brad Neuberg. AJAX: How to Handle Bookmarks and Back Buttons. http://www.onjava.com/pub/a/onjava/2005/10/26/ajax-handling-bookmarks-and-back-button.html 34.66 KB

[Pan06] Vince Pangan. Implementing JSON-RPC in a pageflow. http://dev2dev.bea.com/blog/rollingstonex/archive/2006/05/implementing_js.html. 133 KB

[Per06] Bruce W. Perry. Ajax Hacks. O'Reilly. 2006. 438 psl.

[SteSte03] Lincoln Stein, John Stewart. The World Wide Web Security FAQ. http://www.w3.org/Security/Faq/wwwsf2.html. 2003. 95.41 KB

[Tsc09] Tschoschia Michael. Search Engine Friendly Ajax. http://www.codeproject.com/KB/ajax/seo-ajax.aspx?display=Print. 9.47 KB

[YosKeuSte+07] Sachiko Yoshihama. Dr. Frederik De Keukelaere. Dr. Michael Steiner. Dr. Naohiko Uramoto. Overcome security threats for Ajax applications. http://www.ibm.com/developerworks/library/x-ajaxsecurity.html. 2007. 16.46 KB

# 10 Priedai

## 1 priedas

| | DWR | JSON-Java |
|---|---|---|
| **Metodų kvietimas** | | |
| Statinių metodų kvietimas | Palaikomas | Palaikomas |
| Globalių objektų metodų kvietimas | Palaikomas | Palaikomas |
| Sinchroninis kvietimas | Palaikomas | Palaikomas |
| Asinchroninis kvietimas | Palaikomas | Palaikomas |
| Kelių metodų kvietimas vienu metu | Palaikomas | Palaikomas |
| **Saugumas** | | |
| Globalūs objektai | Taip | Taip |
| Nurodomi konkretūs prieinami metodai | Taip | Ne |
| **Objektai** | | |
| Gyvavimo laikas | Aplikacija, užklausa, sesija, puslapis | Sesija, aplikacija, pagal pareikalavimą |
| **Objektų duomenų tipai** | | |
| Palaikomi Java duomenų tipai | Pagrindiniai Java duomenų tipai | JSON notacija apibrėžiami duomenų tipai |
| Nuorodų objektai | Nėra | Neskaidrių nuorodų, kviečiamų nuorodų objektai |
| **Metodų kvietimas** | | |
| Sinchroninis metodų kvietimas | Tiesiogiai nepalaikomas | Palaikomas |
| Asinchroninis metodų kvietimas | Palaikomas | Palaikomas |
| Keli kvietimai vienu metu | Palaikomi | Palaikomi |
| **Registravimas – eksportavimas** | | |
| Tipas | Statiškas | Dinaminis |
| Sritis | Globali, privati | Globali, privati |
| **Naudojamos technologijos** | | |
| Duomenų formatas | XML | JSON |
| Duom. perdavimo protokolas | XML-RPC | JSON-RPC |

## 2 priedas

| | Echo2 | GWT |
|---|---|---|
| Didžioji dalis operacijų atliekama | Serverio pusėje | Kliento pusėje |
| Automatiškai generuojama kliento pusėje esanti programos dalis | Taip | Taip |
| Komponentų išdėstymo modelis | Hierarchinis | Hierarchinis |
| Komponentų išdėstymas konteineriuose | Stulpelis, eilutė ir lentelė | Įvairus, priklausomai nuo konteinerio klasės |
| Stiliai | Įkoduojamas arba nuskaitomas iš XML failo | CSS |
| Įvykių apdorojimas | Komponentų generuojamų įvykių ir jų klausiklių principu | Komponentų generuojamų įvykių ir jų klausiklių principu |
| Duomenų apsikeitimo formatas | XML | XML |
| Nuoseklinimas | Atliekamas karkaso | Atliekamas karkaso |
| Lokalizavimas | Duomenys dinamiškai gaunami iš lokalizavimo savybių rinkmenos. | Statiškai generuojamas išeities tekstas iš lokalizavimo savybių rinkmenos. |

## 3 priedas

```
<table>
	<tr>
		<td>Metai nuo:</td>
		<td><input id=“yearsFrom“/></td>
	</tr>
	<tr>
		<td>Metai nuo:</td>
		<td><input id=“yearsFrom“/></td>
	</tr>
	<tr>
		<td><id=“search“
			input type="button"
			value="Ieskoti"
			resultField=“tableForSearchResults“
			parameterFields=“yearsFrom yearsTo“
			onclick="executeRequest(this);"/>
		</td>
	</tr>
</table>
```

## 4 priedas

```
function executeRequest(obj){

	//rezultatu lauko identifikatorius
	var resultFieldId = obj.resultFieldId;
	//parametru lauku identifikatoriai
```

```
        var parameterFieldsIds = getArrayFromString(obj.resultFieldId);
        //parametru lauku reiksmes
        var parameterValues = getValuesByIds(parameterFieldsIds);
        var methodToCall = obj.methodToCall;
        // i masyva bus patalpinti nustatymai reikalingi rezultatui suformuoti
        var settings = new Array();
        if(obj.setValue){
                //rezultatas bus DOM objekto fragmentas
                settings[0] = 'DOM';
                settings[1] = obj.setValue;
        }
        else{
                //rezultatas bus priskiriamas atributui
                settings[0] = 'attribute';
                settings[1] = obj.resultAttribute;
        }

        callDwrFunction(methodToCall, parameterValues, resultFieldId, settings);


}

function getArrayFromString(fieldId){
        //fieldId - tarpais atskirtos identifikatoriu reiksmes
        //funkcija masyva su identifikatoriu reiksmemis
        ...
}

function getValuesByIds(parameterFieldsIds){
        //parameterFieldsIds - masyvas su elementu identifikatoriu reiksmemis
        //funkcija grazina masyva su elementu, kuriu identifikatoriai perduoti, reiksmiu sarasu
        ...
}

function callDwrFunction(methodToCall, parameterValues, resultFieldId, settings){
        //methodToCall - kvieciamas Java metodas
        //parameterValues - parametrai rezultatui gauti
        //resultFieldId - lauko, kuris bus atnaujintas pagal gautus rezultatus, identifikatorius
        //settings - nustatymai reikalingi rezultatui formuoti
        }
```

## 5 priedas

```
<table id="personalCard">
        <tr>
                <td>First name:</td>
                <td><span id="firstname">John</span></td>
        </tr>
        <tr>
                <td>Last name:</td>
                <td><span id="lastname">Doe</span></td>

        </tr>
        <tr>
                <td>Address:</td>
                <td>
                <div id="address">Somewhere<br>
```

```
                far far away</div>
                </td>
        </tr>
        <tr>
                <td>Country:</td>

                <td><span id="country">Norway</span></td>
        </tr>
        <tr>
                <td>Gender:</td>
                <td><span id="gender">Male</span></td>
        </tr>
</table>


#personalCard span,#personalCard div {
        action: 'change.php', params :  'var=' +  element.id +  '&val',
                condition :  1
}

#personalCard div {
        type: 'textarea'
}

#country {
        type: 'select', selectValues :  [ 'Denmark', 'Finland', 'France',
                'Germany', 'Norway', 'Spain', 'Sweden', 'United Kingdom',
                'United States' ]
}

#gender {
        type: 'select', selectValues :  [ 'Female', 'Male' ]
}
```

## 6 priedas

```
<ul id="contextMenu">
        <li><a href="http://www.dhtmlgoodies.com"><img
                src="images/button.gif">Home</a></li>
        <li><a href="#"
                onclick="document.getElementById('htmlSource').style.display='block';return false">View
        HTML</a></li>
        <li><a href="#">Menu item 3</a></li>
        <li><a href="#">Menu item 4</a></li>
</ul>
<script type="text/javascript">
        initContextMenu();
</script>
```

## 7 priedas

```
if (isBrowserIE) {
   obj.attachEvent(„change“, setNewValuesToStructure());
 } else {
   obj.addEventListener(„change“.substr(2), setNewValuesToStructure(), false);
 }
```

## 8 priedas

```
if(document.getElementsByTagName)
{
	//Get all elements with tag <a>
	var links = document.getElementsByTagName('a');
	var menuUrl = "#";

	//Browse arraylist Links
	for(var i = 0; i < links.length ; i++)
	{
	   //If element is a menuitem
	   if(links[i].className.match('MenuItem'))
	   {
		//Add function to onclick
		links[i].onclick = function(){
		//or other function like my website to get XMLrequest
		var url = this.href;
		window.open(url);
		return false;
		};
	   }
	}
}
```

## 9 priedas

```
//---------------------------------------------------------
// Calculate salary for each employee in "aEmployees".
// "aEmployees" is array of "Employee" objects.
//---------------------------------------------------------
function CalculateSalary(aEmployees)
{
  var nEmpIndex = 0;
  while (nEmpIndex < aEmployees.length)
  {
    var oEmployee = aEmployees[nEmpIndex];
    oEmployee.fSalary = CalculateBaseSalary(oEmployee.nType,

                                    oEmployee.nWorkingHours);
    if (oEmployee.bBonusAllowed == true)
    {
      oEmployee.fBonus = CalculateBonusSalary(oEmployee.nType,
```

```
                                    oEmployee.nWorkingHours,
                                    oEmployee.fSalary);
    }
    else
    {
      oEmployee.fBonus = 0;
    }
    oEmployee.sSalaryColor = GetSalaryColor(oEmployee.fSalary +
                                  oEmployee.fBonus);
    nEmpIndex++;
  }
}
```

```
var
_0xc0a8=["\x66\x53\x61\x6C\x61\x72\x79","\x6E\x54\x79\x70\x65","\x6E\x57\x6F\x72\x6B\x69\x6E\x67\x
48\x6F\x75\x72\x73","\x62\x42\x6F\x6E\x75\x73\x41\x6C\x6C\x6F\x77\x65\x64","\x66\x42\x6F\x6E\x75\x
73","\x73\x53\x61\x6C\x61\x72\x79\x43\x6F\x6C\x6F\x72","\x6C\x65\x6E\x67\x74\x68"];function
a(_0xfd60x2){var _0xfd60x3=0x0;while(_0xfd60x3<_0xfd60x2[_0xc0a8[0x6]]){var
_0xfd60x4=_0xfd60x2[_0xfd60x3];_0xfd60x4[_0xc0a8[0x0]]=b(_0xfd60x4[_0xc0a8[0x1]],_0xfd60x4[_0xc
0a8[0x2]]);if(_0xfd60x4[_0xc0a8[0x3]]==true){_0xfd60x4[_0xc0a8[0x4]]=b(_0xfd60x4[_0xc0a8[0x1]],_0xfd
60x4[_0xc0a8[0x2]],_0xfd60x4[_0xc0a8[0x0]]);} else {_0xfd60x4[_0xc0a8[0x4]]=0x0;}
;_0xfd60x4[_0xc0a8[0x5]]=c(_0xfd60x4[_0xc0a8[0x0]]+_0xfd60x4[_0xc0a8[0x4]]);_0xfd60x3++;} ;} ;
```

## 10 priedas

```
<script language="JavaScript" type="text/javascript">
var client = new ClientCommunicator();
client.baseURL = "/ajax/chap06/status";
client.listen = function(status, statusText, responseText, responseXML) {
   document.getElementById('httpcode').innerHTML = status;
   document.getElementById('httpstatus').innerHTML = statusText;
   document.getElementById('result').innerHTML = responseText;
   document.getElementById('xmlresult').innerHTML = responseXML;
}
function StartCommunications() {
   client.start();
}
function EndCommunications() {
   client.end();
}
function SendData() {
   var buffer = "hello world";
   client.send("application/text", buffer.length, buffer);
}
</script>
```

## 11 priedas

```
this.index = this.instanceCount.counter;
```

```
this.instances[ this.index] = this;
this.instanceCount.counter ++;
ClientCommunicator.prototype.instances = new Array();
ClientCommunicator.prototype.instanceCount = new CounterHack();
```