VILNIAUS UNIVERSITETAS
MATEMATIKOS IR INFORMATIKOS FAKULTETAS
PROGRAMŲ SISTEMŲ KATEDRA

Failų tvarkymas taikant trimatį interfeisą
Using 3D Interface for File Management

Magistro baigiamasis darbas

Atliko: Gediminas Mitrikevičius (parašas)

Darbo vadovas: dr. Kristina Lapin (parašas)

Recenzentas: dr. Vytautas Čyras (parašas)

Vilnius
2009

Santrauka

Naudojant įprastas 2D failų naršykles esant dideliems (>1000) failų medžiams, kyla problemų, kaip patogiai ir greitai tvarkyti bylas. Šio darbo tikslas yra pasiūlyti patogesnį failų pavaizdavimo būdą ir sukurti didelių medžių failų naršyklės 3D vizualizaciją. Jai iškelti panaudojamumo kriterijai, pagal juos apžvelgiant pavaizdavimo būdus pasirinkta kūginių medžių vizualizacija, aprašyti sistemos reikalavimai, pasirinkta technologija ir realizuotas sistemos prototipas. Sukurta sistema FSN tenkina iškeltus kriterijus ir yra tinkama didelių failų medžių pavaizdavimui.

Raktiniai žodžiai: failas, kūginis, naršyklė, trimatis, interfeisas.

Summary

It is not convenient to browse large file systems (more than 1000 files) using ordinary 2D file browsers. The purpose of this paper is to suggest file tree visualization, suitable for large file trees and implement 3D file system browser prototype. According to the needed browser tasks, there was selected cone tree visualization, was defined specification, selected suitable technology and implemented 3D browser prototype. This system implemented all specified tasks and is suitable for large file tree visualization.

Key words: file, cone, browser, 3D, interface.

Turinys

Įvadas

Tyrimo objektas

Šiame darbe tiriami failų sistemų vizualizavimo būdai. Pasirinktas būdas pritaikomas praktikoje aprašant ir sukuriant trimačio (toliau – 3D) naudotojo interfeiso failų sistemos naršyklę „FSN“.

Aktualumas

Efektyvus dokumentų valdymas kompiuteriuose yra viena pagrindinių naudotojo interfeiso problemų daugelį metų [CM+00]. Vienas iš būdų yra naudoti dvimatį (toliau – 2D) išdėstymą. Šis būdas mums yra įprastas, kuomet dokumentui pavaizduoti naudojama piktograma, naudojami katalogai. Tačiau esant labai dideliam duomenų kiekiui, pavyzdžiui, 1000-iui ar daugiau bylų, atrasti norimą yra sudėtinga ir nepatogu. Taip pat ir su kitokio tipo dokumentas, kaip kad internetinėmis nuorodomis. Tiesa, jei naudojami katalogai, jie gali sudaryti medį, kurį galima matyti visą iškart ar tik to katalogo turinį, kuriame esame. Jei medis nėra didelis, problemų nekyla, tačiau jei jis tampa sudėtingas, turi daug atšakų ir daug dokumentų, rasti norimą objektą tikrai problematiška. Taigi, tampa vis svarbiau rasti gerus tos informacijos valdymo būdus. Vienas iš jų yra informacijos vizualizacija. Ji gali būti apibrėžta taip:

“Vizualus abstrakčios informacijos kiekio ir struktūros pateikimas, skirtas palengvinti jos pasisavinimą ir supratimą.“ [And02]

Naudojant tinkamus informacijos vizualizacijos įrankius, įmanoma išgauti bendrą labai didelio informacijos kiekio vaizdą, o prireikus, sklandžiai atrinkti konkrečias dalis, pamatyti informacijos detales. Šie įrankiai naudojasi žmogaus įspūdingomis suvokimo galimybes greitai apžvelgti, atpažinti ir atsiminti paveiksliukus. Žmogaus vizualaus suvokimo sistema gali ypatingai greitai *automatiškai* aptikti šablonus ir dydžio, spalvos, judesio, sudėties pasikeitimus. Interaktyvios informacijos vizualizacijos sistemos gali išnaudoti šį gebėjimą ir sutalpinti didelius kiekius informacijos į lengvai valdomus monitorius. [Nor05]

Tikslai ir uždaviniai

Darbo tikslas yra sukurti didelių medžių failų naršyklės 3D vizualizaciją, tenkinančią pasirinktus panaudojamumo kriterijus. Apžvelgsime hierarchinių duomenų vaizdavimo technologijas ir nustatysime tinkamiausias priemones mūsų tikslui pasiekti.

Apie informacijos vizualizacija galima kalbėti tipų, kaip vaizduojama informacija, terminais. Jie gali būti suklasifikuoti į 6 tipus:

- Linijinis – lentelės, programų išeities tekstas, abėcėliniai sąrašai, chronologiškai surūšiuoti elementai.
- Hierarchinis – medžių struktūra.
- Tinklai – grafai, kaip kad semantiniai tinklai, žiniatinklis.
- Vektoriai – vektoriai sugeneruoja iš elemento turinio, siekiant pavaizduoti jo charakteristikas. Tekstiniuose dokumentuose vektoriai gali išreikšti (raktinių) žodžių pavartojimo dažnumą ir taip pavaizduoti dokumentą.
- Erdvinis – dvimačių ir trimačių duomenų pavaizdavimas: žemėlapiai, namų aukštų planai.

Kadangi mūsų nagrinėjamas objektas - failų sistema yra hierarchinė struktūra, tai toliau nagrinėsime tik šio tipo vizualizacijos priemones.

Apibrėžkime tipines užduotis, naudojamas dirbant su informacijos vizualizacijos sistemomis:

- Peržiūra – visos informacijos vaizdas.
- Priartinimas – norimų elementų padidinimas, išskyrimas.
- Filtravimas – nedominančių elementų atmetimas.
- Detalios informacijos gavimas – pasirink vieno elemento ar elementų grupės detalios informacijos pateikimas.
- Ryšiai – ryšiu tarp elementų peržiūrėjimas
- Istorija – Atšaukti veiksmus, įvykdyti atšauktus veiksmus.
- Išskyrimas – ieškoti tam tikro elementų poaibio.
- Organizuoti – rankiniu būdu tvarkyti elementus, kad būtų lengviau juos vėliau rasti.

Informacijos vizualizacijos įrankių užduotis yra kiek įmanoma padėti naudotojams atliekant šiuos veiksmus. Mūsų sukurtas trimatis prototipas turės tenkinti šiuos kriterijus:

- Vykdyti šias funkcijas: peržiūra, priartinimas, ryšiai, organizavimas.
- Vaizduoti didelės apimties medžius, sudarytus iš >1000 failų bendrai visuose lygiuose. Failų medis turi tilpti matomo programos lango ribose.
- Katalogam ir failam pateikti papildomą statistinę informaciją.

Siekiant sukurti didelių medžių failų naršyklės 3D vizualizaciją turi būti išspręsti šie uždaviniai:

- Išnagrinėti 3D dokumentų valdymo sistemų vaizdavimo būdus, teigiamas ir neigiamas savybes.
- Išnagrinėti šiuo metu naudojamas ir galimas panaudoti technologijas, jų teikiamas galimybes.
- Išnagrinėti jau sukurtas failų sistemų naršykles.
- Remiantis lyginamosios analizės išvadomis, apibrėžti kuriamos dokumentų valdymo sistemos reikalavimus, pateikti interfeiso eskizus.
- Pasirinkti technologiją ir realizuoti sistemos interfeiso prototipą.
- Įvertinti sukurtos sistemos prototipą

Atsižvelgdami į iškeltus kriterijus vertinsime vizualizacijos priemonės tinkamumą mūsų tikslui. Apžvelgsime jau sukurtas 3D failų sistemų naršykles. Jas suskirstysime pagal vizualizavimo būdus. Taip pat pasirinksime technologiją, tinkamą pasirinkto tipo – kūginių medžių – naudotojo interfeisui realizuoti. Aprašysime kuriamą 3D failų sistemos naršyklę „FSN“. Pagal apibrėžtus reikalavimus, naudojant pasirinktą technologiją realizuosime „FSN“ prototipą. Pateiksime jos interfeisų pavyzdžius, aprašysime iškūlusias problemas ir tolimesnius darbus.

Rezultatai

3D interfeisų technologijų teorinė lyginamoji analizė. Atsižvelgiant į vertinimo kriterijus atrinktas tinkamiausias vizualizacijos būdas. Sukurtų 3D failų sistemų apžvalga. 3D interfeisų technologijų teorinė lyginamoji analizė. Atsižvelgiant į vertinimo kriterijus atrinkta tinkamiausia sistemos kūrimui. Aprašyti kuriamos sistemos reikalavimai ir projektiniai sprendimai. Realizuotas bei įvertintas kuriamos sistemos prototipas, aprašytos iškilusios problemos, siūlomi tolimesni darbai.

1. Hierarchijų vizualizacijų apžvalga

Hierarchinės struktūros yra itin paplitusios mūsų informacinėje visuomenėje. Pavyzdžiui, įmonės darbuotojų apskaita arba failai ir katalogai jūsų kompiuteryje. Šios hierarchijos nuolat didėja. Net jūsų asmeniniame paprastame namų kompiuteryje gali būtų tūkstančiai failų. Taigi vaizdinės priemonės yra būtinos, norint šią informaciją paversti prieinama ir suprantama.

Hierarchinis medis yra formuojamas nuo viršaus iki apačios. Vidiniai elementai turi vaikus, o medžio labai yra paskutinieji elementai. Medyje egzistuoja vienetinis kelias nuo viršūnės iki kiekvieno medžio lapo. Pavyzdžiui failų sistemose vidiniai elementai yra katalogai, o lapai – failai. Kiekvienas failas gal būti pasiektas iš sistemos šaknies atidarinėjant katalogus. Dokumentų valdymo sistemose vidiniai elementai gali būti kolekcijos, o lapai – dokumentai. Sprendimų medyje vidiniai elementai yra pasirinkimai, lapai – sprendimai. [WW+99]

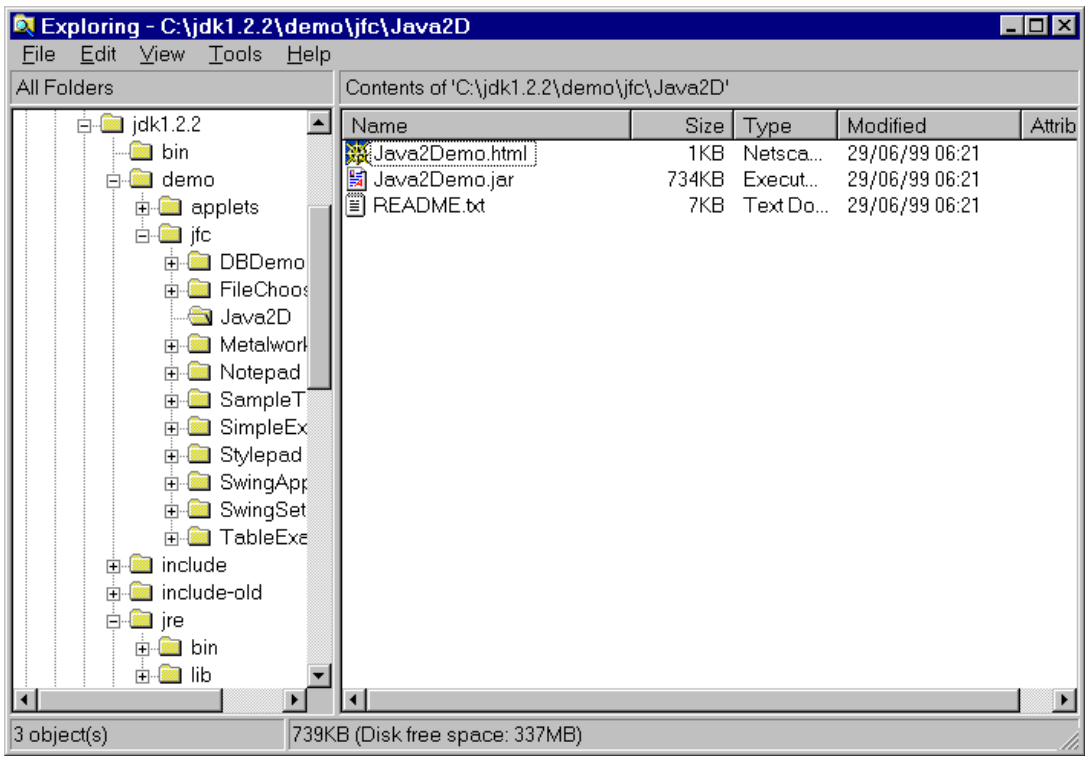
Tradicinės vizualizacijos technologijos, tokios kaip minėta medžių diagrama ar schemos, yra geriausias pasirinkimas vaizduoti nedideles, kompaktiškas hierarchijas, kurios gali būti nupieštos ant vien popieriaus lapo ar pavaizduotos kompiuterio ekrane, viename lange. Jei hierarchija yra didelė, vaizduojama informacija tampa sunkiai suprantama.

Apžvelgsime, kokiais būdais galima pavaizduoti hierarchines struktūras trimatėje erdvėje.

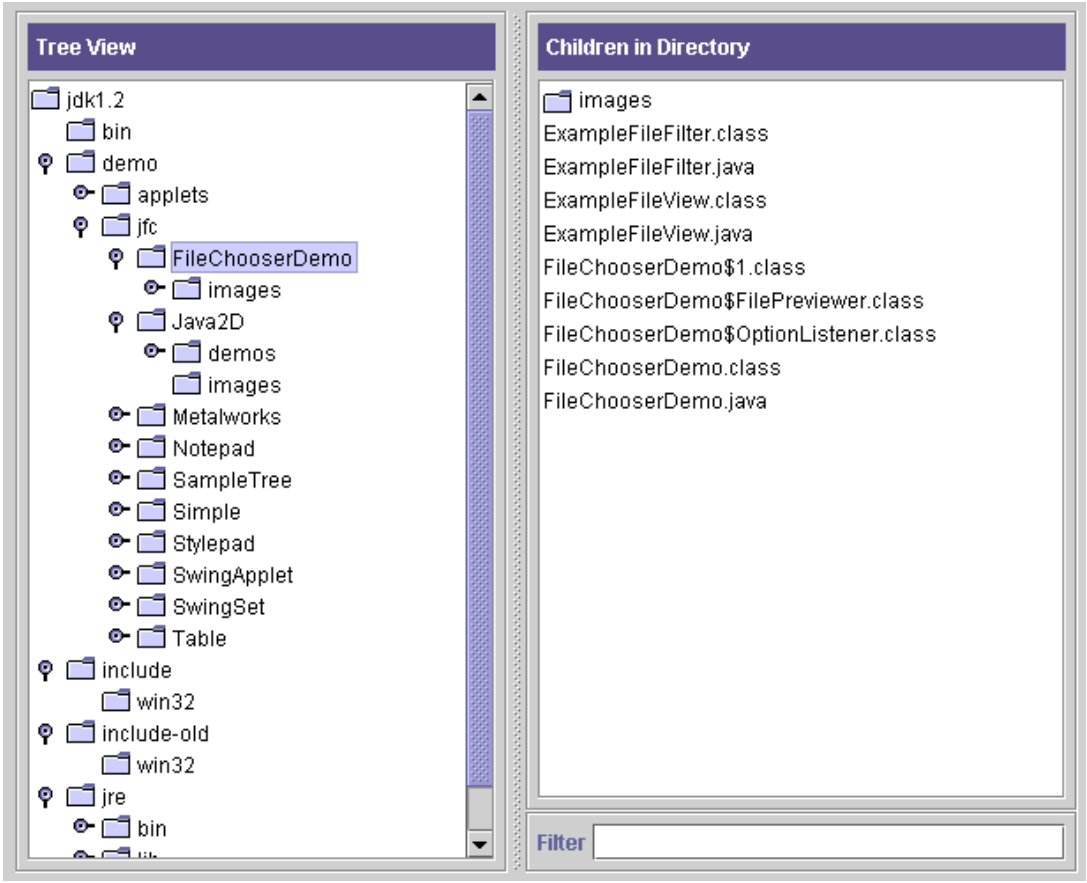
1.1. Medžių naršyklės

Tradicinės medžių naršyklės, tokios kaip Macintosh File Finder ar Windows Explorer hierarchijos pavaizdavimui naudoja eskizinį metodą ir yra naudojamos kasdien milijonų žmonių. Jos yra lengvos realizuoti ir tinka net tekstinėms sistemoms. Tipiškai, ne visi vidiniai katalogai yra iškart išskleidžiami. Vietoj to, naudotojas pats išskleidžia ir sutraukia pomedžius ir pats nusprendžia, kokiu gilumu ir platumu hierarchija yra parodoma.

Kaip matome pirmame paveikslėlyje (žr. 1 pav.), kairėje Windows Explorer langa pusėje yra pavaizduojama medžio struktūra, o dešinėje parodoma pasirinkto elemento sudėtis. Kairioji pusė pateikia kontekstą, tu tarpu dešinioji – dėmesio objektus. Java Swing JTree komponentas, pavaizduotas antrame paveikslėlyje (žr. 2 pav.) yra labai panašus.



1 pav. Windows Explorer. Bendras katalogų vaizdas kairėje, o turinys dešinėje.

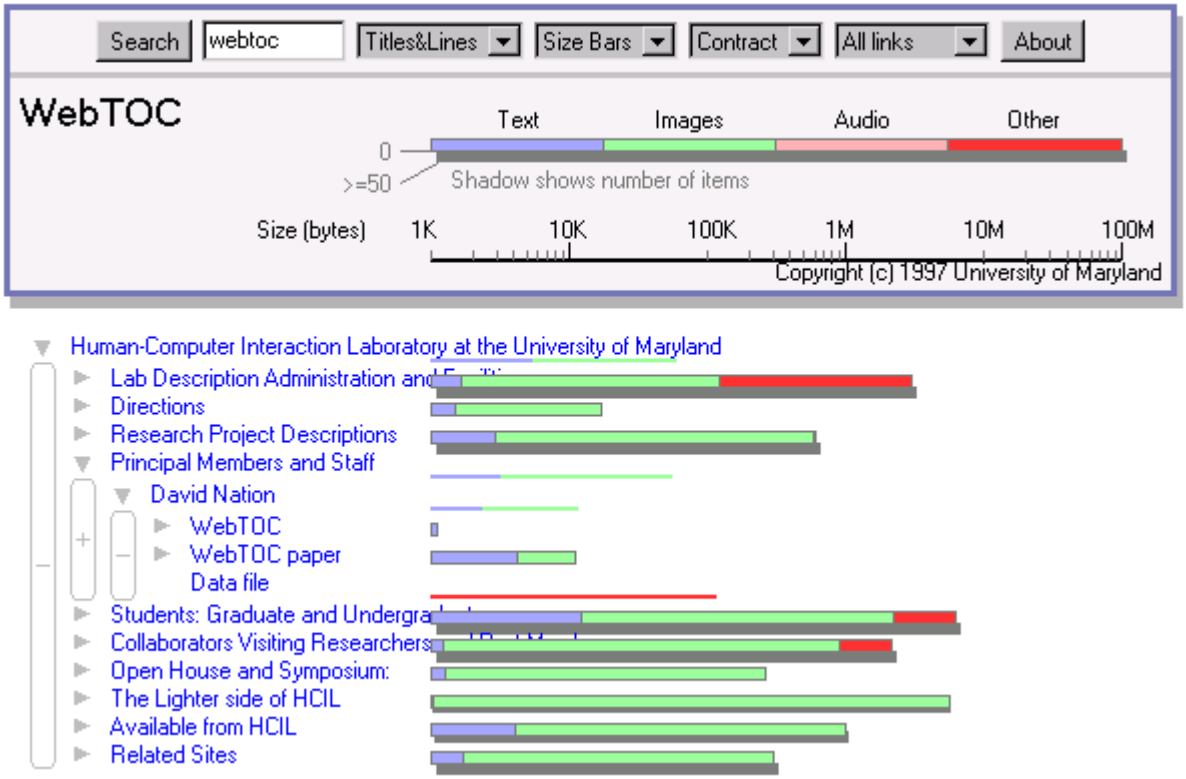


2 pav. Java Swing JTree naršyklė.

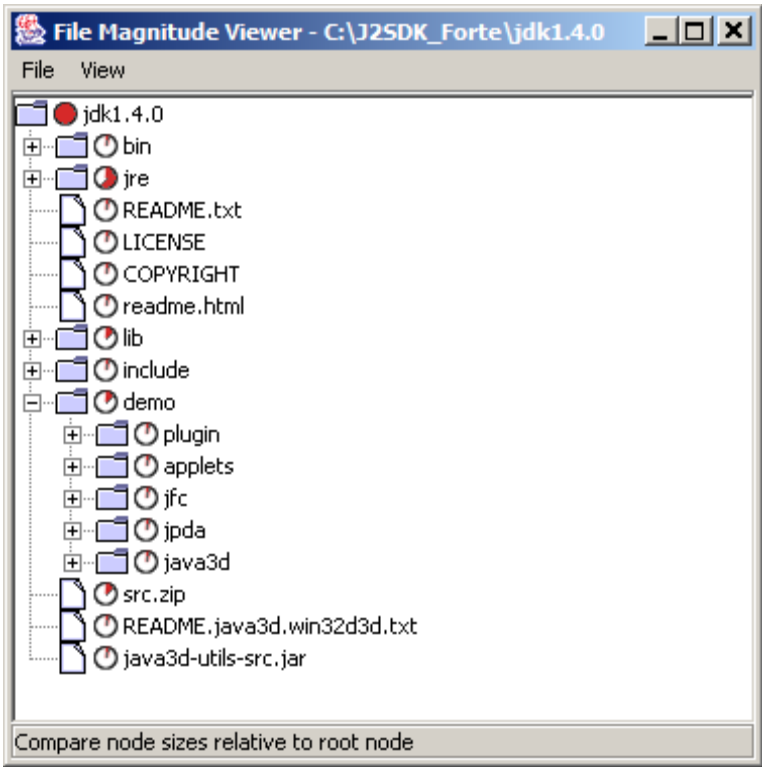
WebTOC leidžia naršyti internetinį puslapį panaudodamas iš anksto sugeneruotą hierarchinę struktūrą (turinį). [NPM+97] Kai hierarchija yra sugeneruojama, išsaugoma ir statistinė informacija apie failų tipus ir dydžius. Ji naudojama pateikti kiekvieno katalogo turinio bendrą vaizdą. Nuspalvintos linijos parodo įvairių failų tipų proporcijas, šešėlis – failų kiekį. (žr. 3 pav.) Taigi turime tradicinį medį su papildoma statistine informacija.

Šią idėją realizuoja ir Java programa File Magnitude Viewer. [SC+01] Ji skenuoja failų sistemą ir prie kiekvieno katalogo pateikia pyrago tipo paveiksliuką, kuris vaizduoja susijusio katalogo ar failo dydį. (žr. 4 pav.)

Nors pastaruosius du įrankius galima sąlyginai vadinti trimačiais ir jie pateikia statistinę informaciją apie katalogus, jie netenkina kitų mūsų išvardintų kriterijų. Jie nėra tinkami dirbti su dideliais medžiais.



3 pav. WebTOC. Eilutė į dešinę 3 nuo kiekvienos šakos parodo nuotraukų, tekstinių, audio ir kitų tipų failų proporcijas. Šešėlis parodo failų kiekį.



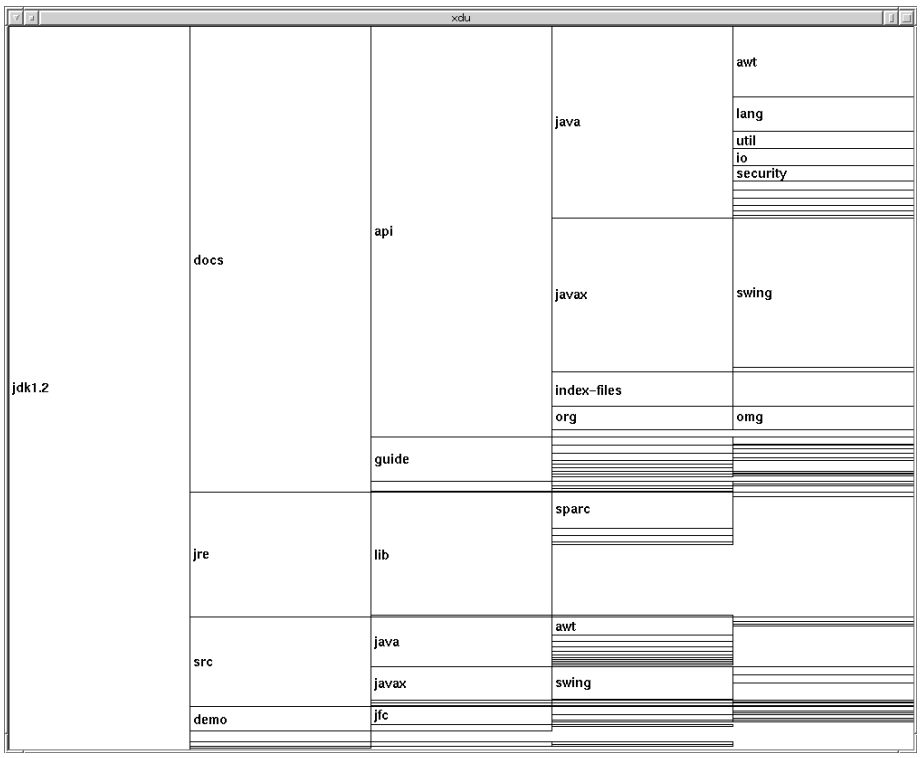
4 pav. File Magnitude Viewer pateikia pyrago formos paveiksliuką, parodantį kiekvieno susijusio pomedžio dydį.

1.2. Medžių žemėlapiai

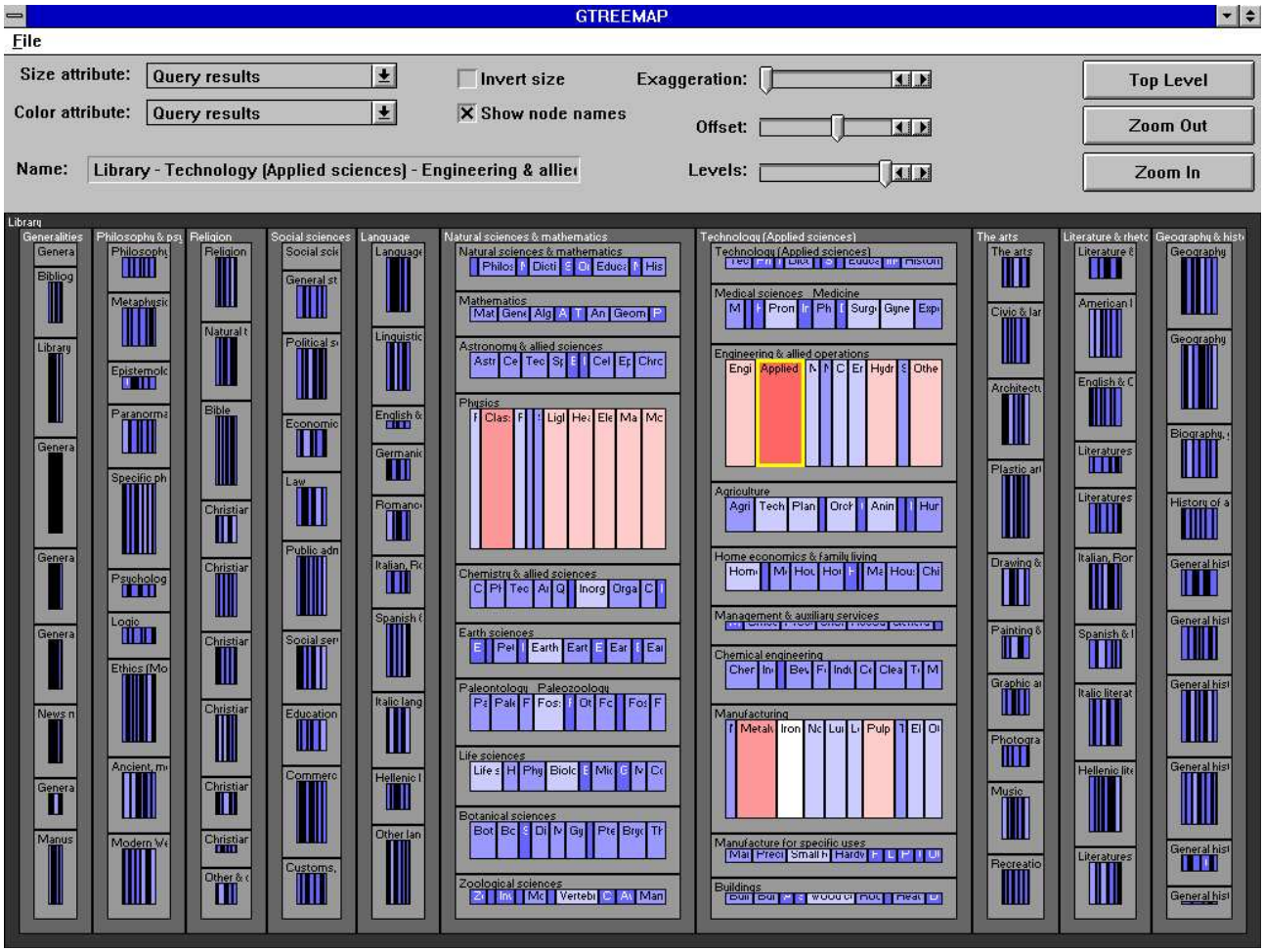
Xdu yra X yra kietojo disko tvarkymo įrankis, kuris grafiškai parodo, kaip panaudotas kietasis diskas Unix failu sistemoje. [Dyk91] Stačiakampiai yra piešiami iš kairės į dešinę leidžiantis medžiu žemyn. (žr. 5 pav.) Vertikali erdvė kiekviename lygyje proporcingai atvaizduoja kiekvienos direktorijos dydį.

Medžių žemėlapiai (žr. 6 pav.) – tai hierarchinėm struktūroms skirta erdvės užpildymo technologija. [JS+91] Tradiciniai medžių žemėlapiai laisvą ekrano vietą dalina vertikaliai i horizontaliai. Kiekvienos juostos dydis yra proporcingas jos *svoriui* – tipiškai lemtų skaičiui ar dydžiui joje. Pagrindinė tokios technologijos problema yra tai, kad stačiakampiai dažnai sumažėja iki itin siaurų ir žemų. Turgaus žemėlapiai (angl. Market map) yra medžių žemėlapių plėtinys, skirtas išvengti šios problemos. [Wat99] Tai „kvadratiška“ versija, kurios generavimui naudojami euristiniai metodai. (žr. 7 pav.)

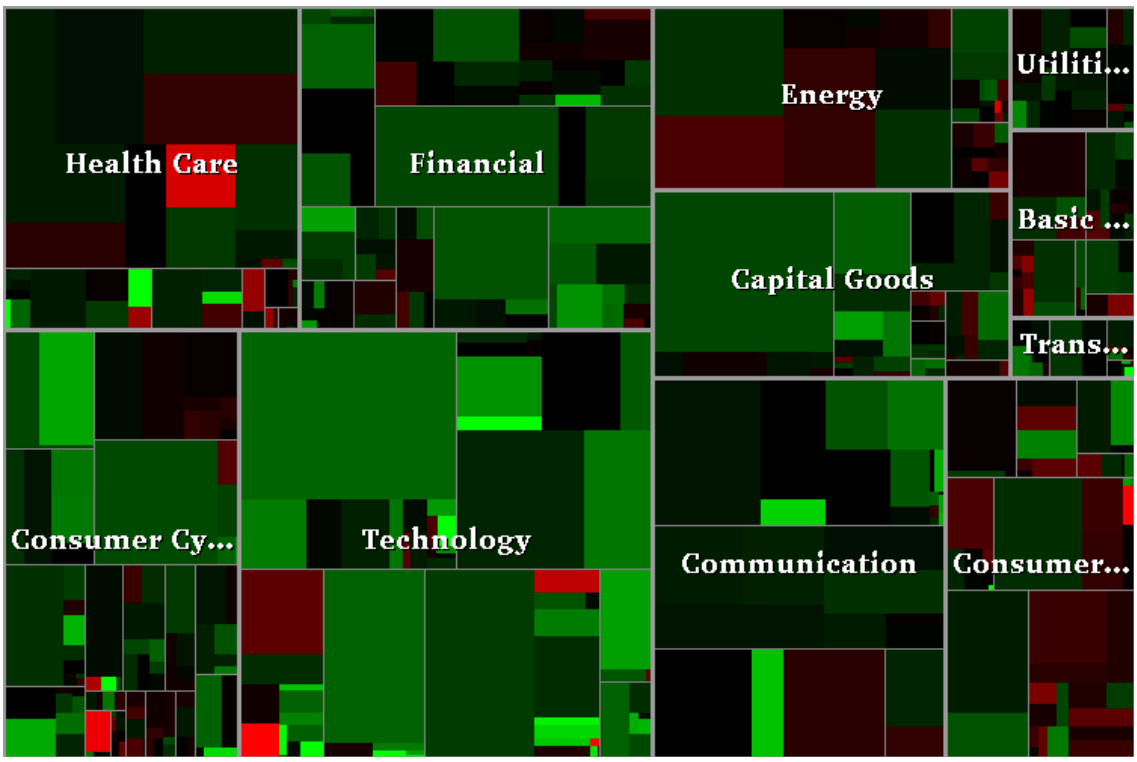
Šio tipo vaizdavimas taip pat netenkina mūsų iškeltų kriterijų. Esant dideliems medžiams tampa nebepatogu dirbti.



5 pav. Xdu naršyklės Java JDK katalogo vizualizacija.



6 pav. Tokio tipo medžių žemėlapiai dažnai naudojami bibliotekose.



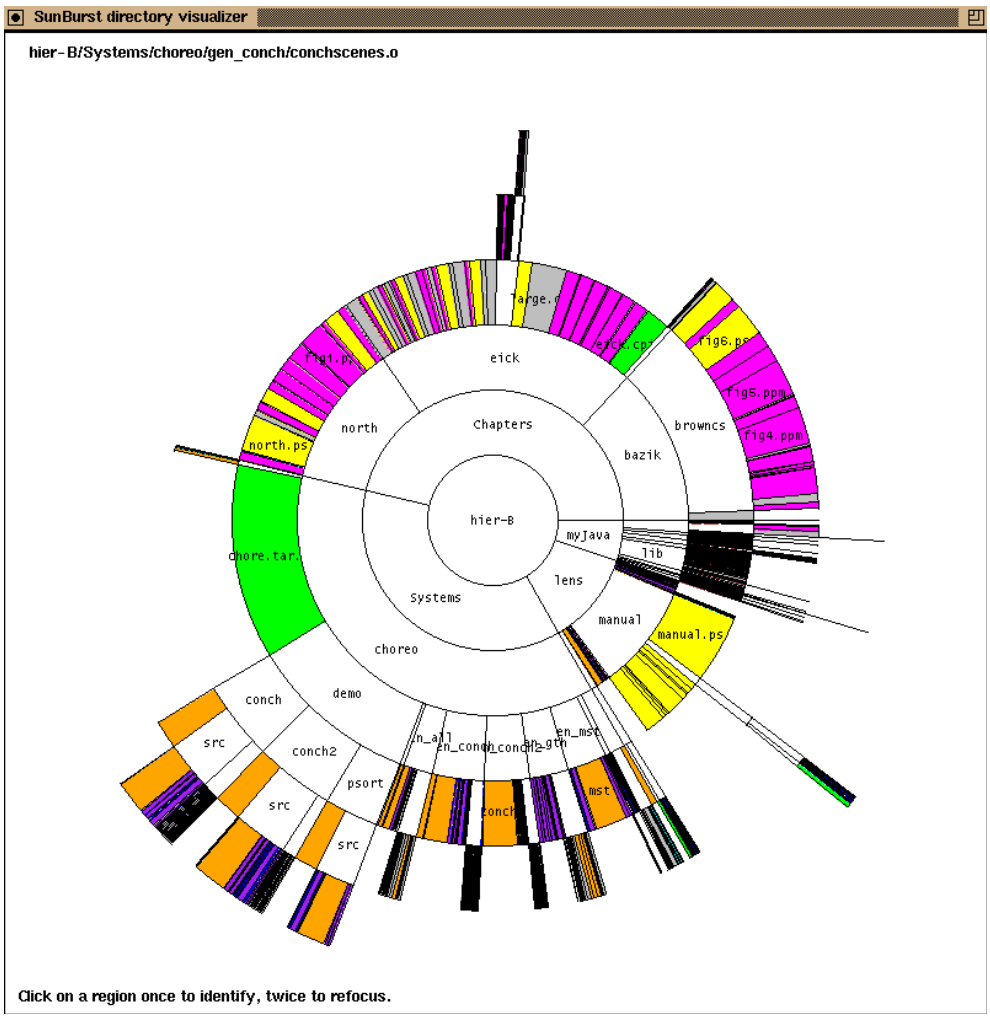
7 pav. Biržos vizualizacija turgaus medžiu. Krentančios akcijos vaizduojamos raudonai, o kylančios – žaliai.

1.3. Spinduliniai pavaizdavimo būdai

Sunburst [SZ+00] naudoja skritulinę diagramą įgyvendinant tą pačią informacijos dalinimo idėją, minėtą praėjusiame skyriuje (žr. 8 pav.)

CyberGeo Maps tinklalapio puslapių išdėstymui naudoja žvaigždžių ir galaktikos metaforas. [HFB98] Rankiniu būdu redaguota hierarchija yra suformuojama atsižvelgiant į katalogų struktūrą tinklalapyje. Šakninis katalogas atitinka saulę sistemos vidury. Taškai (žvaigždės), vaizduojančios puslapius, patalpinamos orbitose aplink saulę, priklausomai nuo to, keliomis nuorodomis jie yra nutolę nuo šaknies. Siekiant išvengti susidūrimų, naudojama paprasta maišo funkcija, kuri remiasi katalogų pavadinimais, ir atsitiktine tvarka išdėlioja žvaigždes į tam tikras orbitas. Hierarchijos lygiai suformuoja koncentrinius žiedus apie šaknį, tačiau gretimos žvaigždės nebūtinai yra panašaus turinio. (žr. 9 pav.)

Šis būdas netenkina mūsų iškeltų kriterijų. Nors ir gali būti pateikiama statistinė informacija, tačiau esant dideliems medžiams tai taip pat nėra patogus būdas.



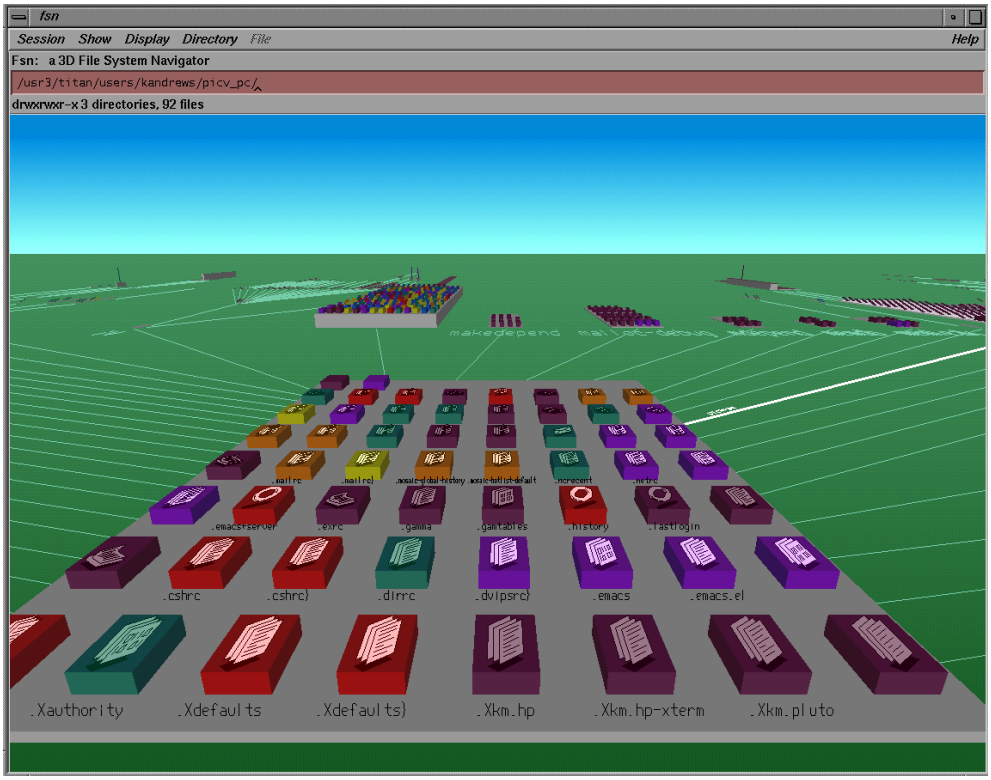
8 pav. Skritulinė diagrama, turinti įvairių interaktyvių valdymo būdų.

9 pav. CyberGeo žemėlapis – saulės sistema.

1.4. Peizažai

File System Navigator (FSN) yra trimačio peizažo failų sistemos vizualizacija. [TS+92] Failai padėti ant pilkų pjedestalų, kiti katalogai atitraukti toliau į foną (žr. 10 pav.) Ikonų aukšti proporcingas jų dydžiui baitais. Spalva nurodo failo amžių. Naudotojas gali laisvai naršyt po peizažą, be to, gali būti pateiktas ir bendras žemėlapis. Taip pat galima atlikti paieškas, rasti failai peizaže išryškinami apšvietimu. Šis vaizdavimo būdas sulaukė pasaulini dėmesio po pasirodymo filme „Jurų periodo parkas“.

Šis būdas netenkina vienintelio mūsų iškelto kriterijaus – esant dideliems medžiams matoma tik dalis jo, taigi tampa nebepatogu naršyti po jį.

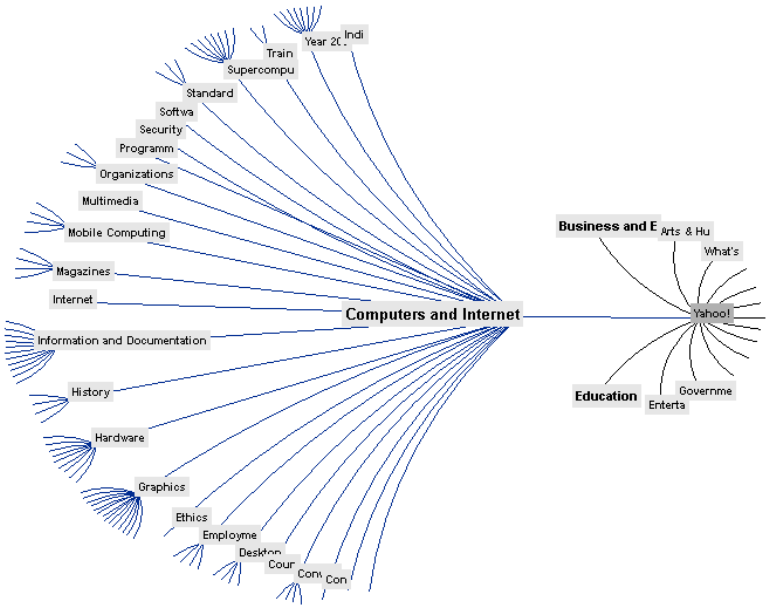


10 pav. FSN peizažinė vizualizacija. Priekyje ant pjedestalo iškeli failai, o tolumoje matomi kiti katalogą ir linijos iki jų.

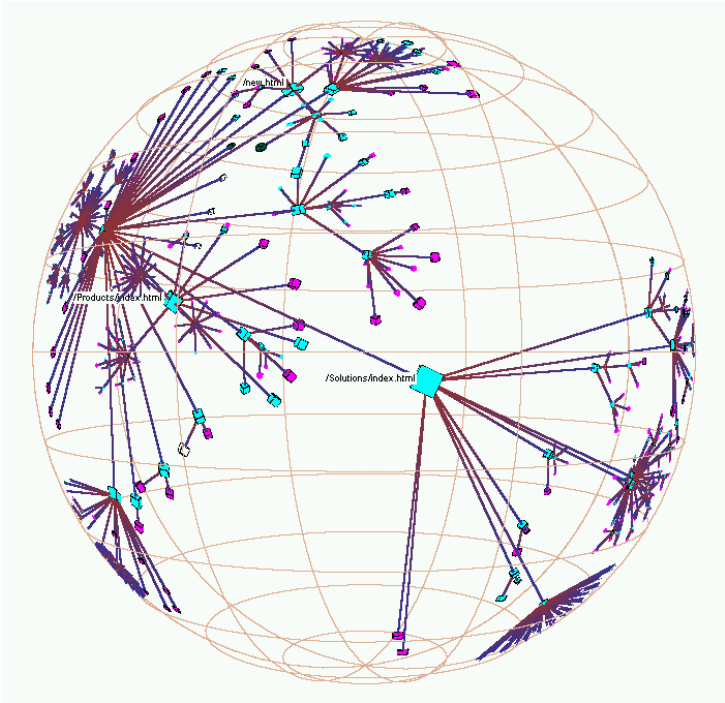
1.5. Hiperbolinės naršyklės

Hyperbolic Browser yra viena žinomiausių informacijos vizualizacijos technologijų. [LR+94] Naudodama hiperbolinį išdėstymą, ši naršyklė suteikia erdvę užpildančią, papildomą informaciją suteikiančią vizualizaciją. (žr. 11 pav.) Pradinis išdėstymas yra padaromas ant hiperbolinės plokštumos. Pagrindinis to privalumas yra ai, kad į kiekvieną pusę ji suteikia neribotą kiekį erdvės, taigi visada yra pavaizduojama visa hierarchija - kiekviena vaikas savo vaikus įspraudžia į tolstančią begalinę erdvę. Panašiai kaip ir dvimatė

hiperbolinė naršyklė, taip ir trimatė išnaudoja begalinės erdvės išdėstymą, tačiau vietoj apskritimo, ji išdėsto elementus sferoje. (žr. 12 pav.)



11 pav. Hiperbolinės naršyklės visada rodo visą hierarchiją. Tačiau kraštiniai elementai tampa tokie maži, kad net nepastebimi.



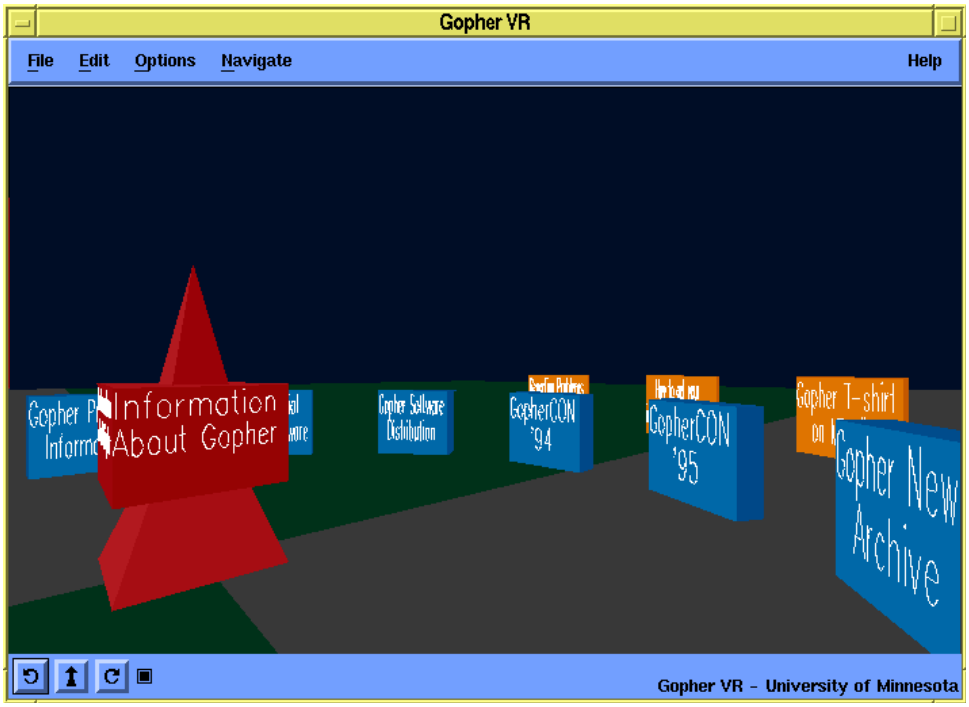
12 pav. Trimatė hiperbolinė naršyklė.

1.6. Kiti bandymai

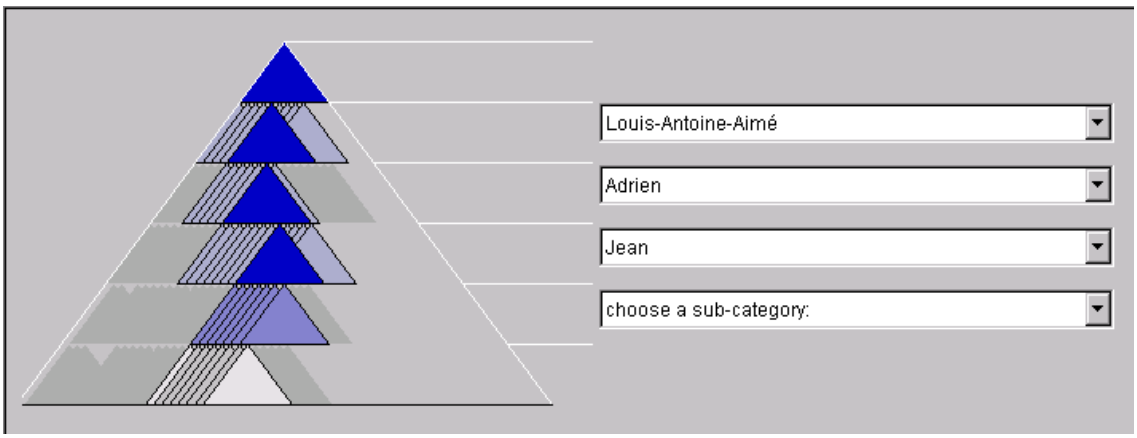
GopherVR – tai peizažo tipo naršyklė, vaizduojanti individualius hierarchijos lygius. [ME+95] Elementai (paieškos rezultatai ar ranka koreguota hierarchija) išdėstoma spirale,

besisukančia nuo centro. Tai kad vienu metu galima matyti tiek vieną hierarchijos lygį ir yra pagrindinė šios naršyklės problema. (žr. 13 pav.)

Cheops sistema, siekdama kompaktiškai pavaizduoti hierarchiją, naudoja vieną ant kito sudėtus trikampius. [BPV96] Vieno lygio vaikai yra suspaudžiami į trikampių eilutę, siekiant sutaupyti vietos. Naršymas vyksta nu viršaus į apačią. Vienu metu galima pasirinkti tik aktyvios šakos vaikys (pavaizduoti tamsesne mėlyna spalva). (žr. 14 pav.)



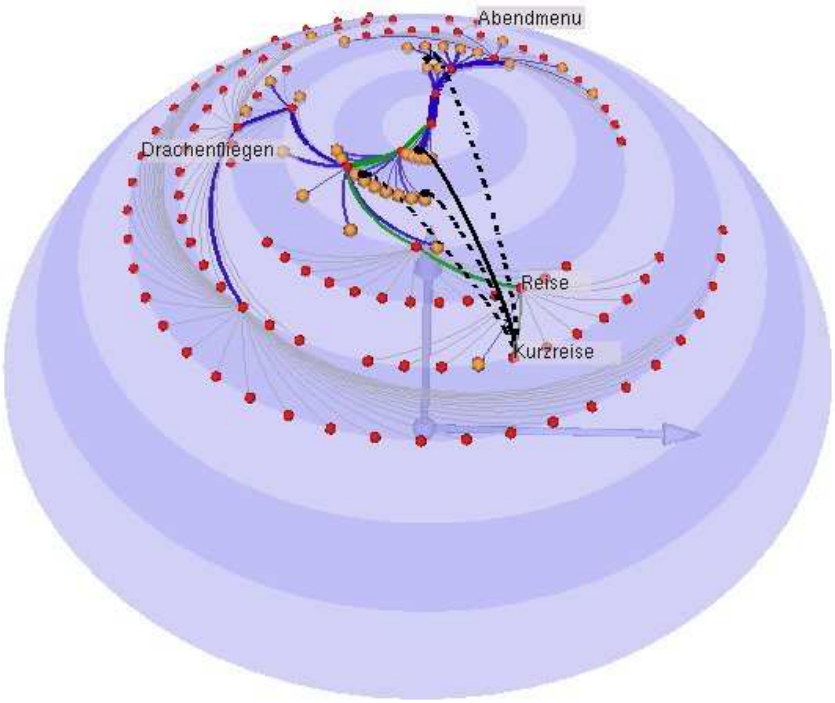
13 pav. GopherVR naršyklėje centrinė piramidė nurodo lygio, kuriame esame, pavadinimą. Paspaudus ant jos, naudotojos grąžinamas į aukštesnį lygį.



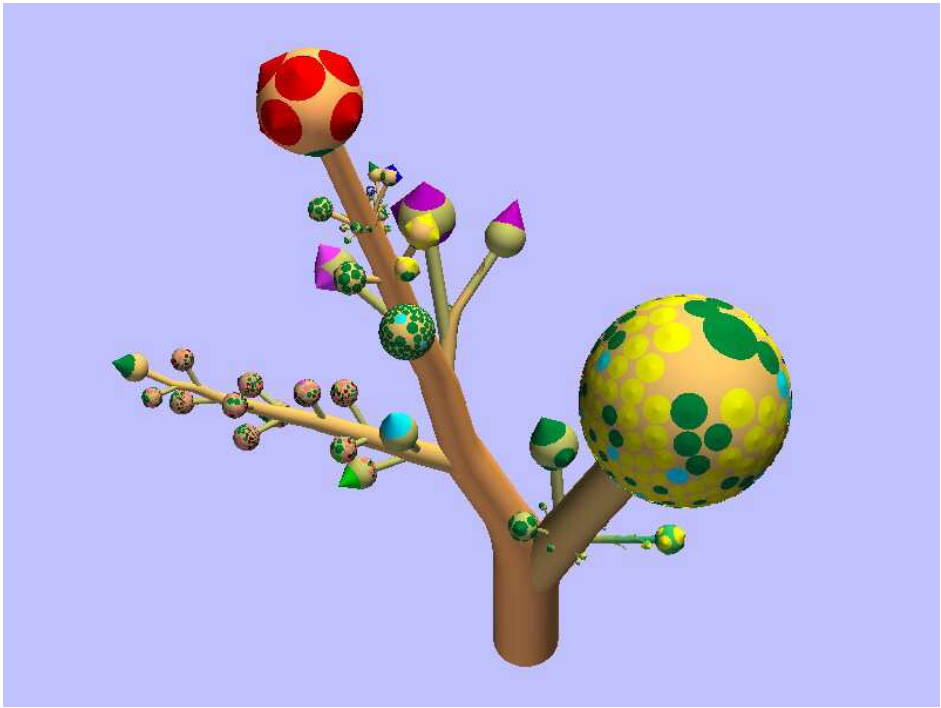
14 pav. Cheops naudoja trikampius, sukrautus vienas ant kito. Tik tamsiai mėlynai pavaizduoti trikampiai – pasirinktos šakos vaikai, gali būti pasirinkti.

Magic Eye View hierarchija pirmiausiai suformuojama dvimatėje erdvėje, naudojant klasikinius algoritmus, o tada ji pritaikoma pusrutuliui. [Bur99] Naršymas vyksta animuotai. Tai labai panašu į hiperbolines naršykles, tačiau čia neišnaudojamos hiperbolės teikiamos galimybės. (žr. 15 pav.)

Botaninė vizualizacija hierarchiją transformuoja į geometrinį šakų ir lapų modelį. [KWW01] Dėl geresnio turinio supratimo, ilgas šakas vaizduoja kaip vaisius, Kūrius paspaudus jie išskleidžiami. (žr. 16 pav.)



15 pav. Magic Eye View dvimatis medis uždėtas ant pusrutulio. Praplečia dvimačio medžio galimybes, tačiau neišnaudoja hiperbolės teikiamų privalumų.



16 pav. Botaninė vizualizacija – hierarchijos lapai vaizduojami kaip vaisiai.

1.7. Duomenų kalnas

Duomenų kalnas – naujoviškas dokumentų valdymo interfeisas sukurtas specialiai išnaudoti žmogaus erdvinio suvokimo galimybes. [RCL+98] Paveiksliuke pateiktas prototipas (žr. 17 pav.) yra alternatyva dabartinių internetinių naršyklių adresynų (angl. bookmarks, favorites) mechanizmams, taigi kalbėdami apie jo objektus, turėsime galvoje internetinius puslapius, tačiau reikia pastebėti, kad šis vizualizavimo būdas gali būti taikomas įvairiems duomenims tvarkyti.

Tikrame pasaulyje erdvinė atmintis dažnai padeda mum surasti daiktus. Pavyzdžiui padėję popieriaus lapą į krūvą ant savo stalo mes ilgai atsiminsime, kur daugmaž ji yra. Ši idėja ir pritaikyta duomenų kalnui.

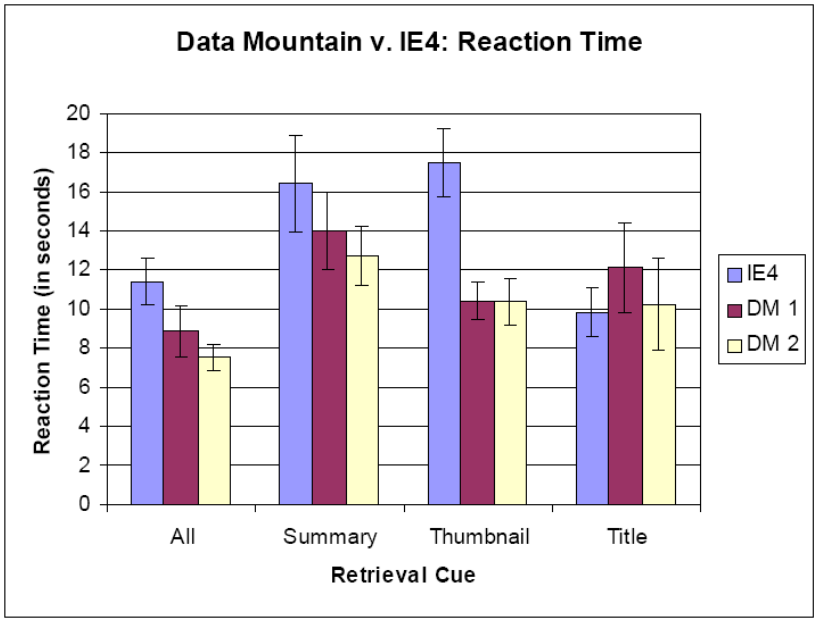
Kai puslapis išsaugomas, pirma jis parodomas ekrano centre. Naudotojas gali patalpinti puslapį nuvilkdamas jį į normą vietą pelės kursoriumi. Kai puslapis yra velkamas, kitos puslapių ikonos pasitraukia jam iš kelio, kad nekas nebūtų uždengiama ir naudotojas suprastų aplinkinių puslapių padėtį. Kai puslapis patalpinamas į vietą, norint į vėl iškvietė į ekrano priekį tereikia vienąkart spragtelti pele.

Pirminė prototipo idėja, kaip jau minėjome, buvo išnaudoti žmogaus suvokimo galimybes. Mūsų galimybė suprasti paprastą trimatį gylį pagal objektų tarpusavio vaizdinius ryšius įgalina patalpinti puslapių ikonas tolumoje ir suprasti jų erdvinius ryšius net negalvojant. Prototipas naudoja vientisą plokštuminį paviršių, pakreiptą 65 laipsniais (nuo to kilo „kalno“ pavadinimas) nuspalvintą tam tikromis spalvomis, kad būtų sustiprintas erdvės jausmas. Siekiant sustiprinti erdvinį jausmą, buvo panaudoti ir garsiniai signalai. Visi jie reaguoja į objektų judesius. Pavyzdžiui velkant puslapį naudotojas išgirsta tam tikrą garsą, kuris keičiasi priklausomai nuo vilkimo greičio. Pasitraukiant kitiems puslapiams iš kelio taip pat sukeliamas atitinkamas švilptelėjimo garsas.

Sukūrus šį prototipą buvo atlikti jo tyrimai. [AGL+03] Dvi duomenų kalno versijos (antrojoje buvo pagerintas automatinis puslapių išdėliojimas ir jį animuotas judėjimas) buvo lyginamos tarpusavyje ir su Internet Explorer 4. Buvo lyginamas atliekamų operacijų greitis, naudojimo patogumas, pripratimas. Rezultatų diagrama aiškiai parodo, kad tvarkant dokumentus naudojant duomenų kalną operacijos atliekamos kur kas greičiau. (žr. 18 pav.)

Šis vizualizacijos būdas tenkina beveik visus mūsų kriterijus. Reiktų jį papildyti tik statistine informacija, leisti duomenų grupėms suteikti vardus. Tačiau toks animuotas interfeisas yra sudėtingas realizuoti, taigi mes jo nesirenkame. Labiausiai atitikęs mūsų kriterijus buvo paskutinis apžvelgiamas būdas – kūginiai medžiai.

17 pav. Duomenų kalnas.



18 pav. Reakcijos laiko palyginimas atliekant veiksmus su ikonomis, jų grupėmis.

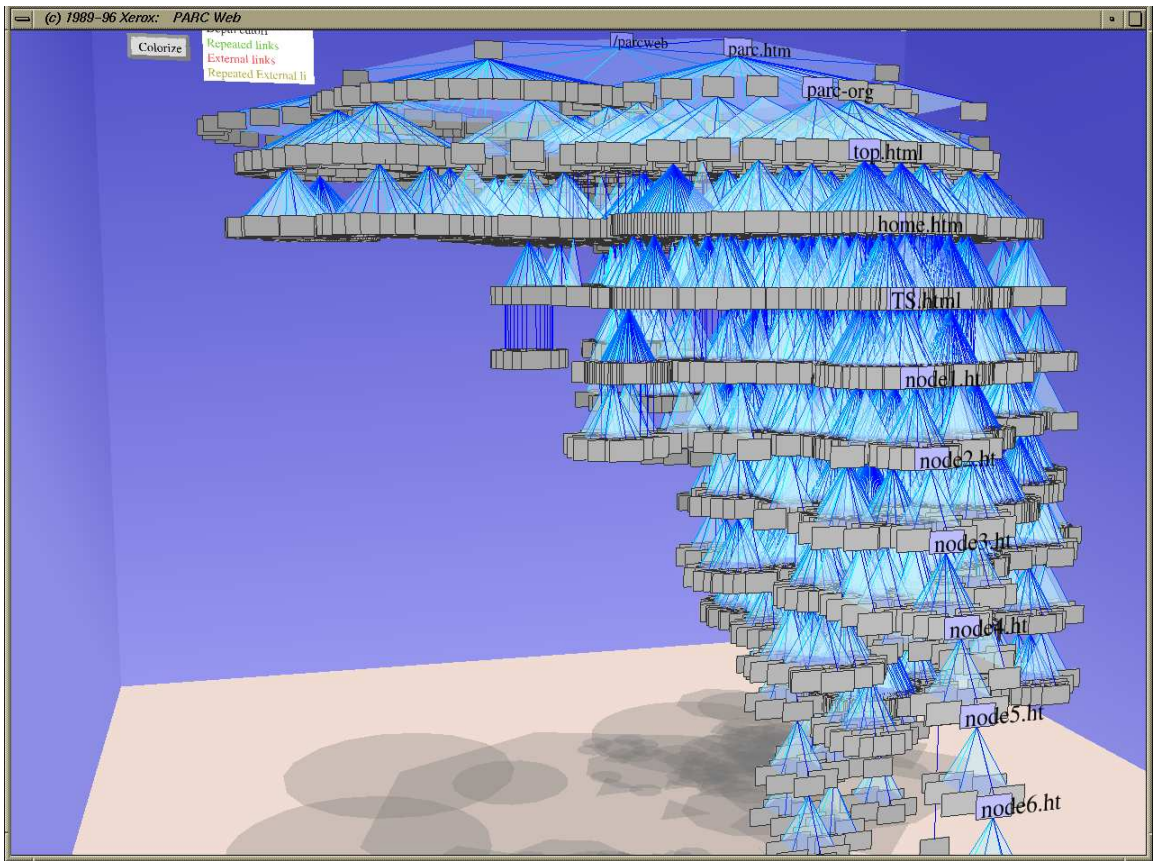
1.8. Kūginiai medžiai

Kūginis medis [MHG06] yra klasikinis trimatis kūginis hierarchijos pavaizdavimas. Galima tiek vertikalus, tiek horizontalus išdėstymas. Pastarojo atveju patogiau surašyti elementų pavadinimus. Hierarchijos šaknis patalpinama prie virtualaus kambario lubų, vaikai piešiami kaip nedideli stačiakampiai, linijomis sujungti su savo tėvu. Piešiant tokiu būdu gaunamas erdvinis daugianaris hierarchinis medis. (žr. 19 ir 20 pav.)

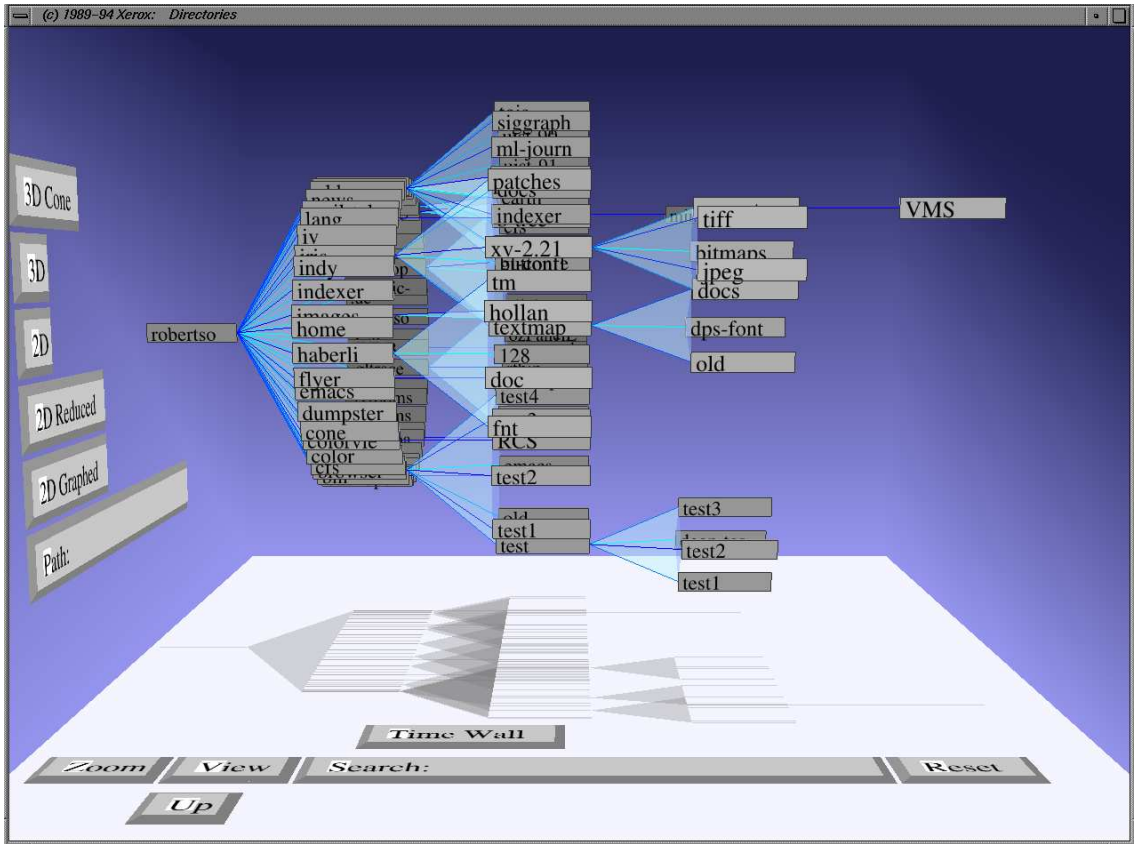
Pasirinkus norimą elementą medis automatiškai pasukamas taip, kad tas pasirinktas elementas ir kiekvienas elementas nuo pasirinktojo iki hierarchijos šaknies būtų atvedami į priekį ir apšviečiami. Medžio sukimas atliekamas naudojant trumpiausio sukimo kelią, taikant įvairius algoritmus. Kiekvienas pometis pasukamas taip, kad reikiamas elementas atsidurtų priekyje. Pats sukimo veiksmas atliekamas ne momentiškai, o palaipsniui ir animuotai, kad žmogus negalvodamas akimis galėtų sekti objektų judėjimą. Taip po kiekvieno pasikeitimo nereikia iš naujo įsiminti, kur kas yra, tai automatiškai atlieka žmogaus erdvinio suvokimo sistema. Tipiškai visa transformacija atliekama maždaug per sekundę, jei ji būtų atliekama, žmogui prireiktų nuo kelių, iki keliasdešimt sekundžių vien tam, kad suprastų, kur kas pasislinko. Kadangi stačiakampiuose visas tekstas dažniausiai netelpa, jis pilnas rodomas tik pasirinkto elemento kelio stačiakampiuose. Be to šį kelią galima išryškinti įvairiai, nebūtinai jį apšviečiant. Pavyzdžiui, galima pakeisti elementų spalvas. Be to, ant apatinės plokštės naudojami šešėliai tik padeda suvokti medžio gylį

Šis trimatis išdėstymas gali būti perkeltas ir į dvimatę plokštumą, tačiau medis gautųsi labai platus ir paprasčiausiai netilptų ekrane. Naudotojui reiktų slinktuku judinti vaizdą ekrane. Trimatis išdėstymas būtent tai ir išsprendžia. Yra išnaudojamas trimatės erdvės gylis ir į ą patį ekraną sutalpinama kur kas daugiau informacijos.

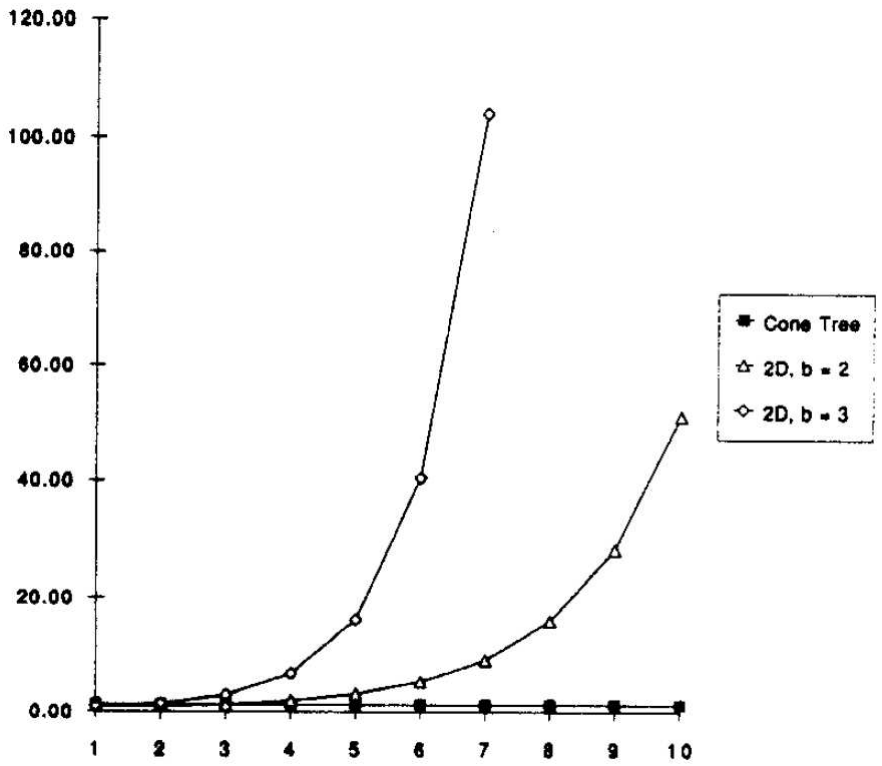
Norint įvertinti kūginio medžio pranašumą prieš dvimačiu medžiu, reiktų įvertinti jų žvilgsnio kampą (angl. aspect ratio). Mum nesvarbu elementų skaičius, svarbu kiek šakų turi medžiai. X ašis atspindi medžio šakų skaičiaus faktorių B, o Y ašis – hierarchinių lygių skaičių L. Žvilgsnio kampo formulė yra B^{L-1}/L. [RMC91] Kaip matome iš diagramos, paprasto dvimačio medžio žvilgsnio kampas didėja eksponentiškai, tuo tarpu trimatis medis lengvai talpina 10 hierarchinių lygių. Tačiau, žinoma, ir jie turi tam tikras ribas. Šioje diagramoje pavaizduoti du dvimačiai medžiai, su didesniu ir mažesniu šakų skaičiumi. (žr. 21 pav.)



19 pav. Kūginio tipo medis. Pomedžiai gali būti interaktyviai atidaryti ir uždaryti. Kai pasirenkamas elementas, visas medis pasisuka taip, kad norimas katalogas ar failas atsidurtų priekyje.



20 pav. Failų sistemos kūginis medis. Atkreipkite dėmesį į šešėlius. Jie padeda suvokti medžio gilumą.



21 pav. Rutuliukais ir trikampiukais pažymėti grafikai atitinka dvimačio medžio žvilgsnio kampo kitimą. Kvadratukais pažymėta kūginių medžių linija vos pakyla virš X ašies.

1.9. Apibendrinimas

Atsižvelgiant į paminėtas kūginio medžio savybes, jis puikiai tinka mūsų iškeltam tikslui realizuoti. Šis vizualizacijos būdas itin tinka dideliems hierarchiniams medžiams pavaizduoti. Jame gali būti atlikta paieška, norimo elemento išryškinimas, aiškiai matomi ryšiai tarp elementų ir jis yra ryškiai pranašesnis už dvimačius medžius. Neišpildomas tik vienas punktas – statistinė informacija. Tačiau tai nėra problema, kadangi kūginį medį galima nesunkiai papildyti 1.1 skyrelyje pateiktomis statistinių medžių savybėmis.

Žemiau pateiktoje lentelėje (žr. 22 lentelę) matomas kūginių medžių pranašumas prieš kitus vizualizacijos būdus. Visi jie, išskyrus duomenų kalną, yra netinkami didelės apimties medžiams vaizduoti. Medžių naršyklės ir spinduliniai vaizdavimo būdai nepateikia elemento priartinimo galimybės. Hiperbolinės naršyklės ir duomenų kalno pavaizdavimo būdas nepateikia statistinės elementų informacijos. Vienintelis pavaizdavimo būdas, tenkinantis visus kriterijus, yra pasirinkta kūginio medžio vizualizacija, papildyta statistine informacija.

	Peržiūra	Priartinimas	Ryšiai	Organizavimas	Didelės apimties medžiai	Statistinė informacija
Medžių naršyklės	+		+	+		+
Medžių žemėlapiai	+	+	+			+
Spinduliai vaizdavimo būdai	+		+	+		+
Peizažai	+	+	+	+		+
Hiperbolinės naršyklės	+	+	+	+		
Duomenų kalnas	+	+	+	+	+	
Kūginiai medžiai	+	+	+	+	+	+
Kiti būdai	+	+	+	+		+

22 lentelė. Hierarchijų vizualizacijų vertinimas.

2. Failų sistemos tvarkymo programos projektavimas

2.1. Sukurtų 3D failų sistemų analizė

Yra sukurta įvairių 3D failų sistemos naršyklių. Analizei atlikti buvo rasta 8 failų sistemos ir 2 interneto naršyklės. Atliekant paiešką, buvo stengiamasi rasti kūginio medžio realizaciją. Atlikdami analizę vertinsime pagal šiuos kriterijus. Sistema turi:

1. Vykdyti šias funkcijas: peržiūra, priartinimas, ryšiai.
2. Vaizduoti didelės apimties medžius, sudarytus iš >1000 failų bendrai visuose lygiuose. Failų medis turi tilpti matomo programos lango ribose.
3. Katalogam ir failam pateikti papildomą statistinę informaciją.

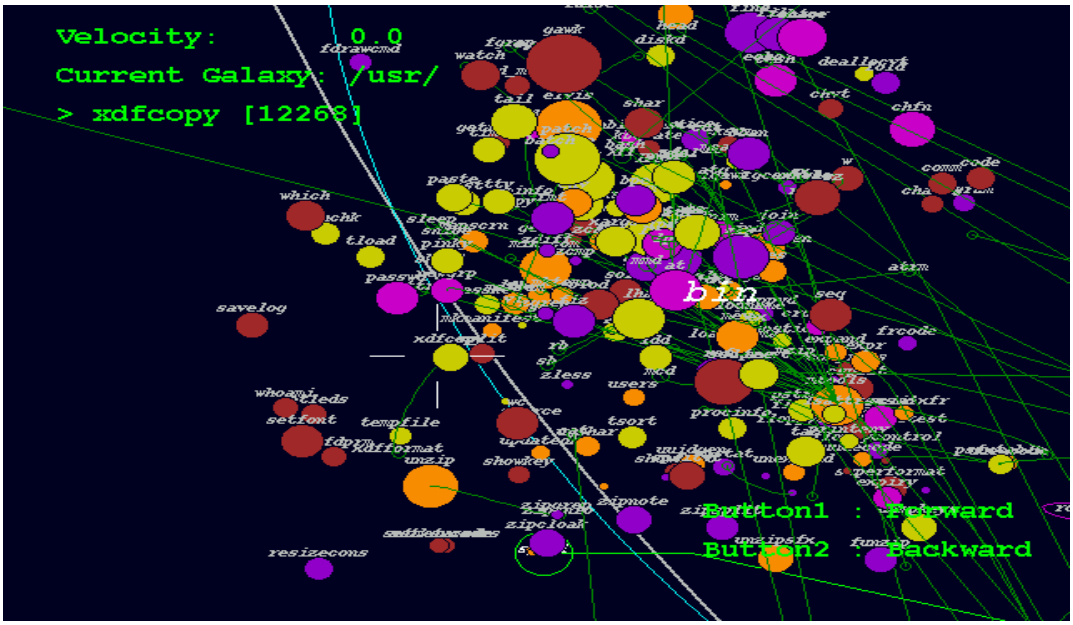
Pateiksime trumpus aprašymus ir iliustracijas:

- Tactile3D – Windows skirta programa, kuri leidžia naudotojui tvarkyti failus ir katalogus 3D erdvėje. Sukuriamas įspūdis, kad dirbama plačioje teritorijoje, kurioje kiekvienas katalogas yra atskiras pasaulis, aplink kuriuos galima klajoti, juose talpinti kitus objektus. Tai peizažo tipo naršyklė. (žr. 23 pav.) Ši sistema turi peržiūros, priartinimo ir ryšių funkcijas. Nors ji pranašesnė už dvimatį pavaizdavimą, tačiau netinkama naudoti didelės apimties medžiams, bei nepateikia statistinės informacijos.

23 pav. Tactile3D failų sistemos naršyklė.

- XCruiser – Linux sistemos3D failų sistemos vizualizavimo įrankis. Leidžia naudotojui skristi per failų sistemą, tarsi tai būtų planetų sistema: katalogai

vaizduojami kaip galaktikos, o failai – kaip planetos (jų dydis priklauso nuo failo dydžio). Ryšiai vaizduojami linijomis. Tai spindulinio vaizdavimo būdo naršyklė. (žr. 24 pav.) Ši sistema netenkina tik vieno kriterijaus – ji netinkama didelių medžių vaizdavimui.

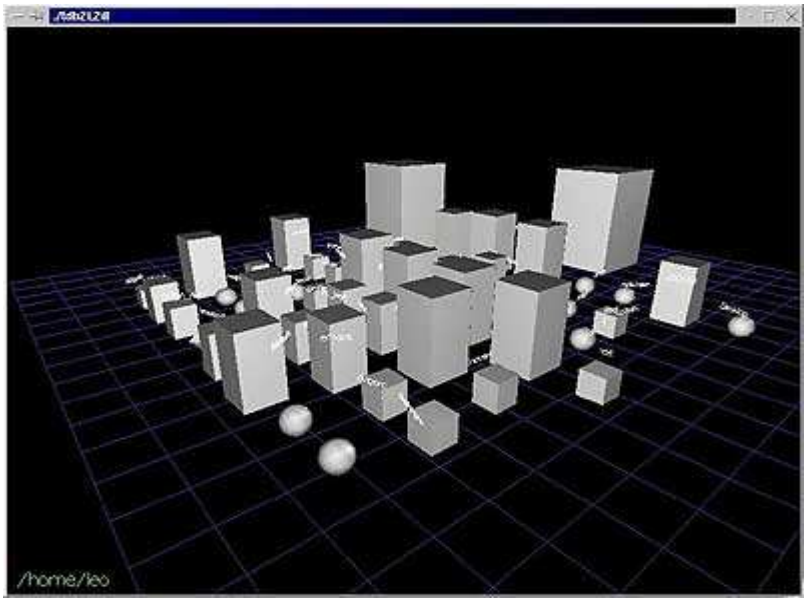


24 pav. Xcruiser failų sistemos naršyklė.

- FSV – Linux skirta vizualizavimo sistema. Failai ir katalogai išdėstomi vienoje platumoje, geometriškai atvaizduojant failų sistemos hierarchiją. Tai peizažo tipo naršyklė. (žr. 25 pav.) Ši sistema taip pat netenkina vienintelio mūsų iškelto kriterijaus – esant dideliems medžiams matoma tik dalis jo, taigi tampa nebepatogu naršyti po jį.

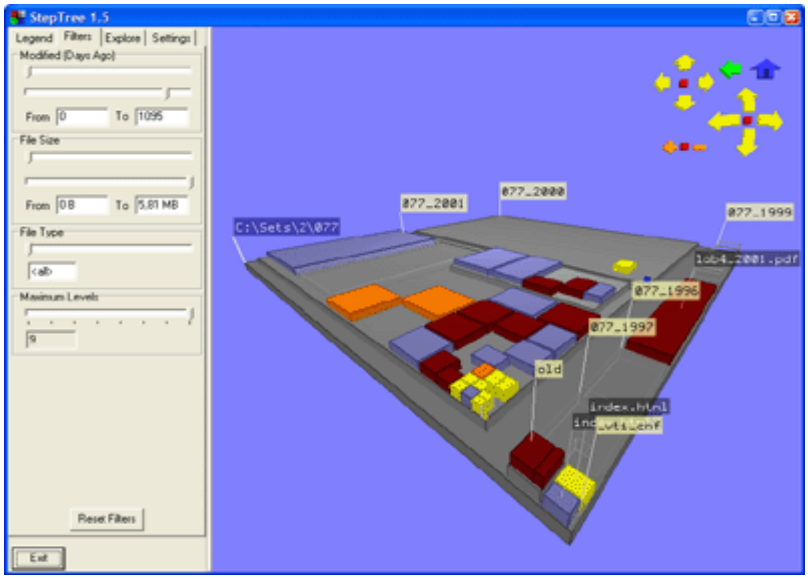
25 pav. FSV failų sistemos naršyklė.

- TDFSB – Linux programa, kuri failus ir katalogus vaizduoja kaip 3D pasaulį, taigi naudotojas gali „pasivaikščioti“ po savo failų sistemą. Tai 3D medžių žemėlapio tipo naršyklė. (žr. 26 pav.) Ji netinkama naudoti dideliems failų medžiams, tačiau pateikiama statistinė informacija ir neįprastas naršymo būdas.



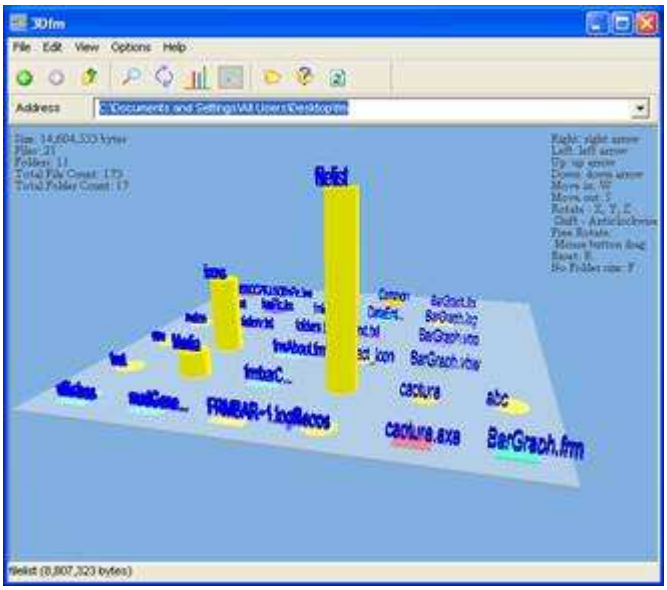
26 pav. TDFSB failų sistemos naršyklė.

- StepTree – Windows 3D sistema, kuri taip pat praplečia medžių žemėlapius ir pateikia trimatį grafinį vaizdą. Hierarchinė sistema vaizduojama kaip dėžutės, sukrautos viena ant kitos vienoje ribotoje platumoje. (žr. 27 pav.) Pavaizduojami ryšiai tarp katalogų, galimas priartinimas ir pateikiama statistinė informacija, tačiau ši sistema taip pat netinkama didelių medžių vaizdavimui.



27 pav. StepTree failų sistemos naršyklė.

- 3DFM – Linux skirta praplėsta medžių žemėlapio tipo sistema, pateikiantį 3D vaizdą. (žr. 28 pav.) Šios programos savybės tokios pat kaip ir StepTree. Ji netinkama didelių medžių vaizdavimui.



28 pav. 3DFM failų sistemos naršyklė.

- 3DNA – Windows sistema, kuri failų sistemą vaizduoja kaip įvairios tematikos pasaulius – povandeninis, kosminė stotis, pobūvių salė. Šį naršyklė nepatenka į nei vieną apibrėžtą kategoriją. (žr. 29 ir 30 pav.) Joje galimas priartinimas, tačiau neaiškiai pateikiami ryšiai tarp katalogų ir būtų sudėtinga pavaizduoti didelius medžius.

29 pav. 3DNA failų sistemos naršyklė.

30 pav. 3DNA failų sistemos naršyklė.

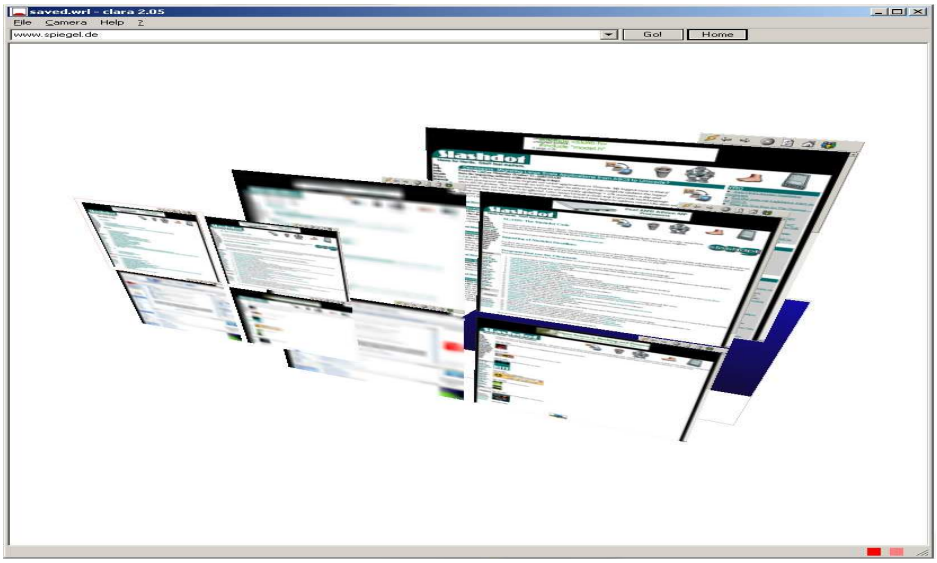
- 3D-Space VFS – MacOS X operacinei sistemai skirta naršyklė, kuri pateikia failų sistemos vaizdą iš paukščio skrydžio. Tai peizažo tipo vizualizacija. (žr. 31 pav.) Ši programa netenkina vienintelio mūsų iškelto kriterijaus – esant dideliems medžiams pavaizdavimas ir naudojimas tampa sudėtingas.

31 pav. 3D-Spave VFS failų sistemos naršyklė.

- SphereXPlorer – Windows 3D interneto naršyklė, nuorodas vaizduojanti kaip atskirus paveikslėlius trimatėje erdvėje. Tai duomenų kalno tipo vizualizacija. (žr. 32 pav.) Kadangi tai yra internetinių nuorodų, tai ji neturi ryšių tarp elementų. Yra netinkama medžio struktūroms vaizduoti ir nepateikia statistinės informacijos.

32 pav. SphereXPlorer interneto naršyklė.

- Clara – Windows 3D interneto naršyklė, leidžianti vaikščioti, skraidyti ir šokinėti po virtualų pasaulį, kur visi objektai yra naudojimui paruošti internetiniai puslapiai, kurios galima skaityti taip, kaip tai būtų daroma 2D erdvėje. Tai duomenų kalno tipo naršyklė. (žr. 33 pav.) Šios programos savybės tokios pat, kaip ir SphereXPlorer.



33 pav. Clara interneto naršyklė.

Žemiau pateikiama visų rezultatų lentelė. (žr. 34 lentelę) Nebuvo rastas nei vienas viešai platinamas kūginio medžio vizualizacijos tipo pavyzdys. Nei viena iš sistemų netenkina visų iškeltų kriterijų. Visos iš rastų naršyklių netinkamos vaizduoti didelius medžius, kai kurios nepateikia statistinės informacijos ar ryšių tarp elementų. Šios savybės yra svarbios failų tvarkymo sistemai, todėl būsimoje „FSN“ 3D failų naršyklėje įgyvendindami iškeltus tikslus išvengsime esamų realizacijų neigiamų savybių.

Vizualizacijos tipas	Pavadinimas	Operacinė sistema	Neįgyvendinti kriterijai	Nuoroda
Medžių naršyklės				
Medžių žemėlapiai	StepTree	Windows	2	http://www.determinate.net/webdata/seg/tdfsb.html
	TDFSB	Linux	2	http://www.determinate.net/webdata/seg/tdfsb.html
	3DFM	Linux	2	http://sourceforge.net/projects/innolab/
Spinduliai vaizdavimo būdai	XCruiser	Linux	2	http://xcruiser.sourceforge.net/
Peizažai	Tactile3D	Windows	2; 3	http://www.tactile3d.com/
	FSV	Linux	2	http://fsv.sourceforge.net/
	3D-Space VFS	Mac OS X	2	http://www.marcmoini.com/f3_en.html
Hiperbolinės naršyklės				
Duomenų kalnas	Sphere XPlorer	Windows	1; 2	http://www.spheresite.com/spherexplorer.html
	Clara	Windows	1; 2	http://www.spatialknowledge.com/projects/clara/
Kūginiai medžiai				
Kiti būdai	3DNA	Windows	1; 2	http://3dna.net/products/desktop.htm

34 lentelė. Sukurtų 3D failų sistemų vertinimas.

2.2. Technologijų parinkimas

Apibrėšime kriterijus, pagal kuriuos rinksimės technologiją:

- Greitas sistemos atsakas, be vizualiai pastebimų trukdžių – dirbant su failų sistema itin svarbus greita sistemos reakciją į naudotojo veiksmus. Kūginio medžio animacija turi vykti be vizualiai pastebimų trukdžių ar pauzių.
- Kuo aukštesnio lygio programavimas – programavimo kalba turi būti objektinė. Tai susiję su technologijos patogumu ir greitesniu įsisavinimu. Taip pat su žemiau esančiu reikalavimu.
- Elementų atitikimas – kūginiai medžiai susideda iš elementų ir ryšių tarp tų elementų. Technologija turi leisti operuoti šiomis esybėmis (objektais), o ne vien tik linijomis ar taškais.

Kadangi technologija turi būti objektinė, tai itin sumažina jų sąrašą. Palyginsime technologijas – Java 3D ir JOGL.

Java 3D yra grafais (angl. scene graph) paremtas 3D programavimo interfeisas (angl. API), skirtas Java platformai. Jis dirba su OpenGL arba Direct3D. Ši technologija pateikia grafinį programavimą kaip objektiškai orientuotą koncepciją. Grafai yra sudaryti kaip medžiai, kuriuose patalpinti elementai, reikalingi pavaizduoti objektus.

JOGL tai biblioteka, kuri Java programavimo kalboje leidžia naudoti OpenGL. Naudojant metodus keliose klasėse galima operuoti OpenGL interfeisu. Šios technologijos abstrakcijos dėka JOGL parašytų sistemų veikimas yra efektyvesnis, palyginus su aukštesnio abstrakcijos lygio Java 3D. Dauguma JOGL kodo yra automatiškai sugeneruojamas, taigi pasikeitimai OpenGL specifikacijoje gali būti iškart pridėti.

Šios technologijos savo galimybėmis yra labai panašios. Žemiau pateikiama kriterijų atitikimo lentelė. (žr. 35 lentelę) Abi aukšto lygio, objektiškai orientuotos, interaktyvios. Tačiau JOGL yra automatiškai atsinaujinanti pagal OpenGL specifikaciją, kai tuo tarpu Java3D atnaujinimas yra vėluojantis, kadangi jis turi būti atliktas žmonių, o ne automatiškai. Labiausiai pastebimas skirtumas – atsako laikas, o tai yra svarbiausias apibrėžtas kriterijus. Taigi „FSN“ 3D failų sistemos naršyklės realizavimui pasirinksime JOGL.

	Greitas sistemos atsakas	Aukšto lygio programavimas	Nuolatinis atnaujinimas
Java 3D		+	
JOGL	+	+	+

35 lentelė. Technologijų palyginimas.

2.3. FSN reikalavimai

Failų sistemos naršyklė FSN yra skirta vidutiniams naudotojams. Tai darbo su kompiuteriu patirties turintys asmenys, kurie nėra profesionalai, tačiau ir ne naujokai. Kompiuterį jie naudoja kasdieniams darbams, tame tarpe ir failų sistemos naršyklę FSN, tačiau ji nėra jų pagrindinis darbo įrankis, su kuriuo jie atliktų kasdienes esmines užduotis. Tokiem vartotojas svarbūs 4 programos aspektai:

- Matomumas, pastebimumas (angl. observability) – naudotojo galimybė įvertinti programos būklę pagal jos pateikiamus vaizdinius duomenis. Tai apima galimybę patogiai naršyti, lengvai rasti ir pasiekti norimus failus ar katalogus ir aiškiai matyti kelią iki jų, medžio dinaminius pasikeitimus.
- Atsakomumas (angl. responsiveness) – programos atsakas naudotojui. Tai failų medžio animuoti veiksmai pasirinkus katalogą ar failą.
- Užduočių atitikimas (angl. task conformance) – sistema turi atlikti visas užduotis, kurių tikisi naudotojas. Tai kelio iki failų ir katalogų pavaizdavimas, galimybė pasirinkti katalogą, jį atidaryti ir pan.
- Atstatymas (angl. recoverability) – galimybė naudotojui atstatyti sistemos būseną įvykus klaidai.

Atsižvelgdami į šias savybes, apibrėšime programą FSN.

2.3.1. Būsimos sistemos įtakojamų asmenų kategorijos

- Pirminiai – kompiuterio naudotojas.
- Antriniai – nėra.
- Tretiniai – nėra.
- Aptarnaujantieji – projektinė grupė.

2.3.2. Panaudojamumo tikslai

2.3.2.1. Diegimo etapo tikslai

- Jei kompiuteryje yra įdiegta Java platforma, sistemos diegimas turi įvykti paspaudžiant nedaugiau kaip 10 klavišų ir neilgiau kaip per 5 minutes.
- Jei kompiuteryje nėra įdiegta Java platforma, diegimas turi įvykti paspaudžiant ne daugiau kaip 25 klavišus ir ne ilgiau nei per 10 minučių.

2.3.2.2. Apmokymo etapo tikslai

- Pirminis naudotojas nėra yra vidutinės patirties, anksčiau naudojęs tokias pat funkcijas atliekančias programas, todėl jis išmoks naudoti 95% sistemos galimybių per 5 minutes.

2.3.2.3. Riboto naudojimo etapo tikslai

- Sistemos sąsajos intuityvumas leis sistemos valdymo komandų aibę įsiminti per 30 minučių.
- Įvykus klaidai, sistema pateiks pranešimą. Gali reikėti sistemą perkrauti.

2.3.2.4. Pilno naudojimo etapo tikslai

- Dėl įgyvendintų interfeiso kriterijų, sistemos naudotojas 10 katalogų gylyje esantį failą, esant išskleistam visam failų medžiui, suras per 5 sekundes.

2.3.3. Naudotojo sąsajos reikalavimai

2.3.3.1. Dalykinės srities metaforos reikalavimai

Sistemoje turi būti naudojama failų sistemos metafora. Sistema pateikia failų valdymo komandas ir rezultatus.

2.3.3.2. Formuluojamos užduotys

2.3.3.2.1. Peržiūra

2.3.3.2.1.1. Sistemos įjungimas [Įjungti sistemą]

2.3.3.2.1.2. Peržiūros rezultatas [Pavaizduotas failų medis]

2.3.3.2.2. Priartinimas

2.3.3.2.2.1. Priartinimas [Pasirinkti elementą]

2.3.3.2.2.2. Priartinimo rezultatas [Elementas ir kelias iki jo yra medžio priekinėje (naudotojo matomoje) dalyje]

2.3.3.2.3. Filtravimas

2.3.3.2.3.1. Filtravimas [Įvesti filtravimo kriterijus]

2.3.3.2.3.2. Filtravimo rezultatas [paieškos rezultatai atitinka kriterijus]

2.3.3.2.4. Detalios informacijas gavimas

2.3.3.2.4.1. Detalios informacijos gavimas [Pasirinkti elementą ir funkciją detaliai informacijai gauti]

2.3.3.2.4.2. Detalios informacijos gavimo rezultatas [Pateikta detali informacija]

2.3.3.2.5. Organizavimas

2.3.3.2.5.1. Organizavimas [kopijuoti, trinti, sukurti failus ir katalogus]

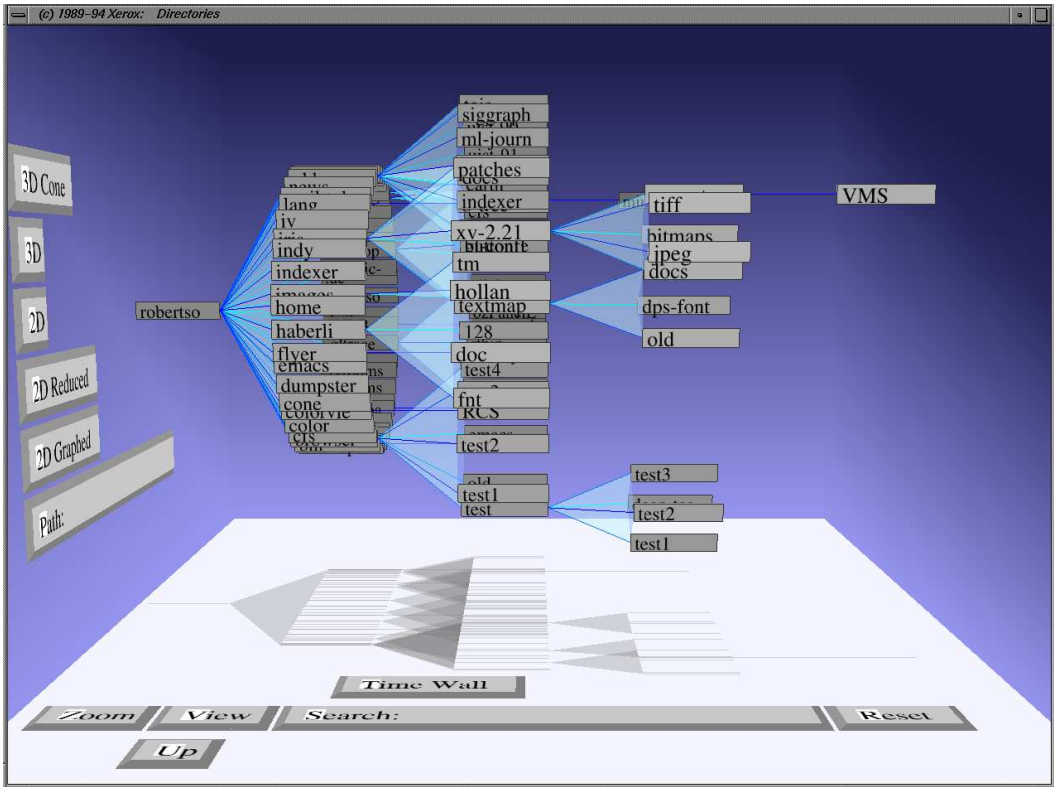
2.3.3.2.5.2. Organizavimo rezultatas [Nukopijuotas, ištrintas, sukurtas failas ar katalogas]

2.3.3.3. Užduočių formulavimo kalbos reikalavimai

Sistemos užduotys turi būti formuluojamos grafinio interfeiso pagalba – jos pateikiamos naudojant dialogo langus, įvedimo elementus, meniu, piktogramas. Užduotys pateikiamos viename lange, bet skirtingose jo dalyse ar kortelės. Informacija įvedama pele ir klaviatūra.

2.3.3.4. Užduočių formulavimo būdo (protokolo) reikalavimai

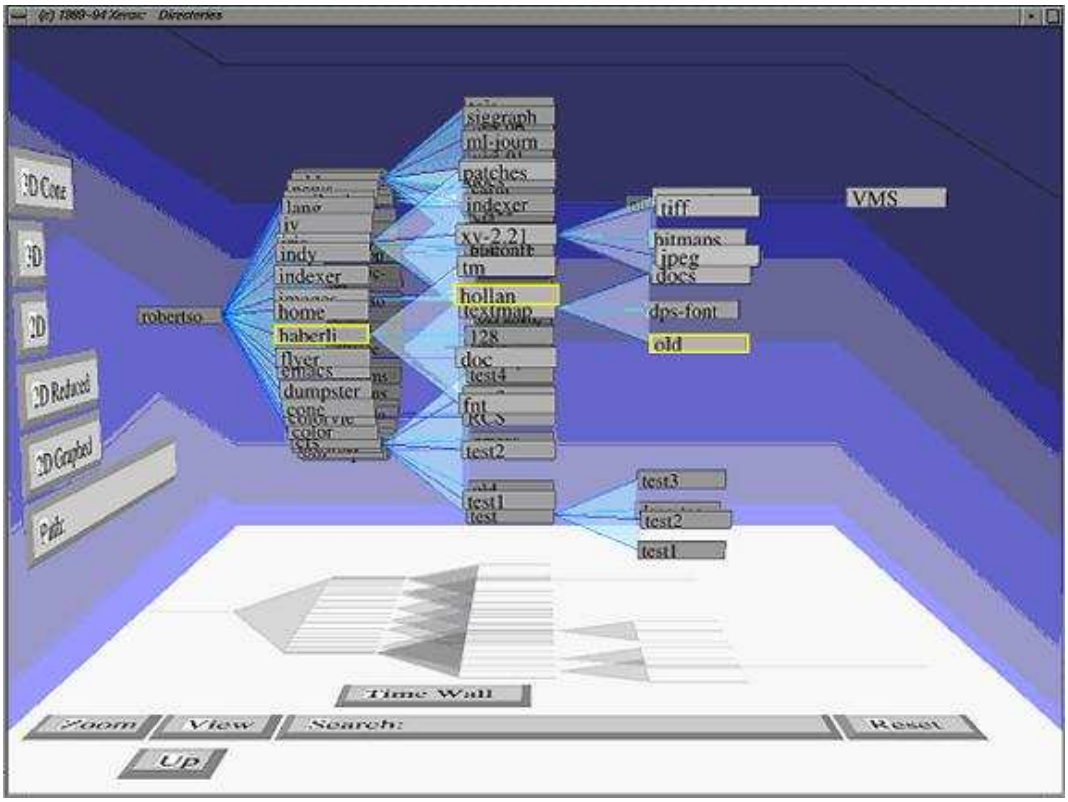
2.3.3.4.1. „Peržiūros“ interfeiso prototipas (žr. 36 pav.)



36 pav. „Peržiūros“ interfeiso prototipas.

Tai visos sistemos langas. Failai ir katalogai vaizduojami kaip kūginis medis, kybantis erdvėje.

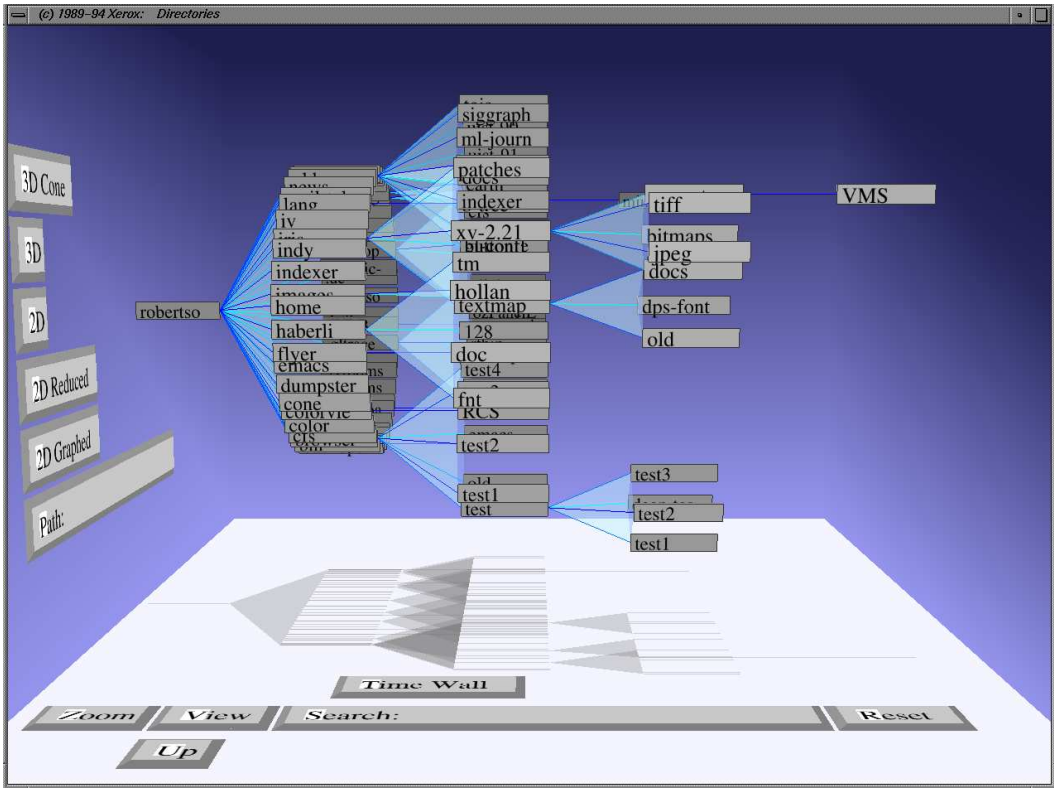
2.3.3.4.2. „Priartinimo“ interfeiso prototipas (žr. 37 pav.)



37 pav. „Priartinimo“ interfeiso prototipas.

Pasirinkus norimą išryškinti elementą, jis ir visi kiti tėviniai jo elementai atsukami į medžio priešakinę dalį ir paryškinami. Paveikslėlyje pateiktas geltonai apvestų elementų pavyzdys: haberli – hollan – old.

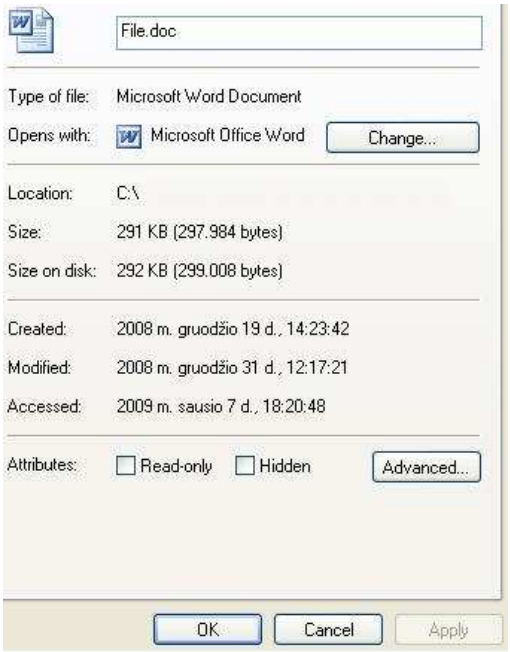
2.3.3.4.3. „Filtravimo“ interfeiso prototipas (žr. 38 pav.)



38 pav. „Filtravimo“ interfeiso prototipas.

Paieška atliekama į erdvės apatinėje platformoje esantį laukelį įrašius kriterijus. Įvykdžius paiešką, pavaizduojamas kriterijus atitinkančių elementų medis.

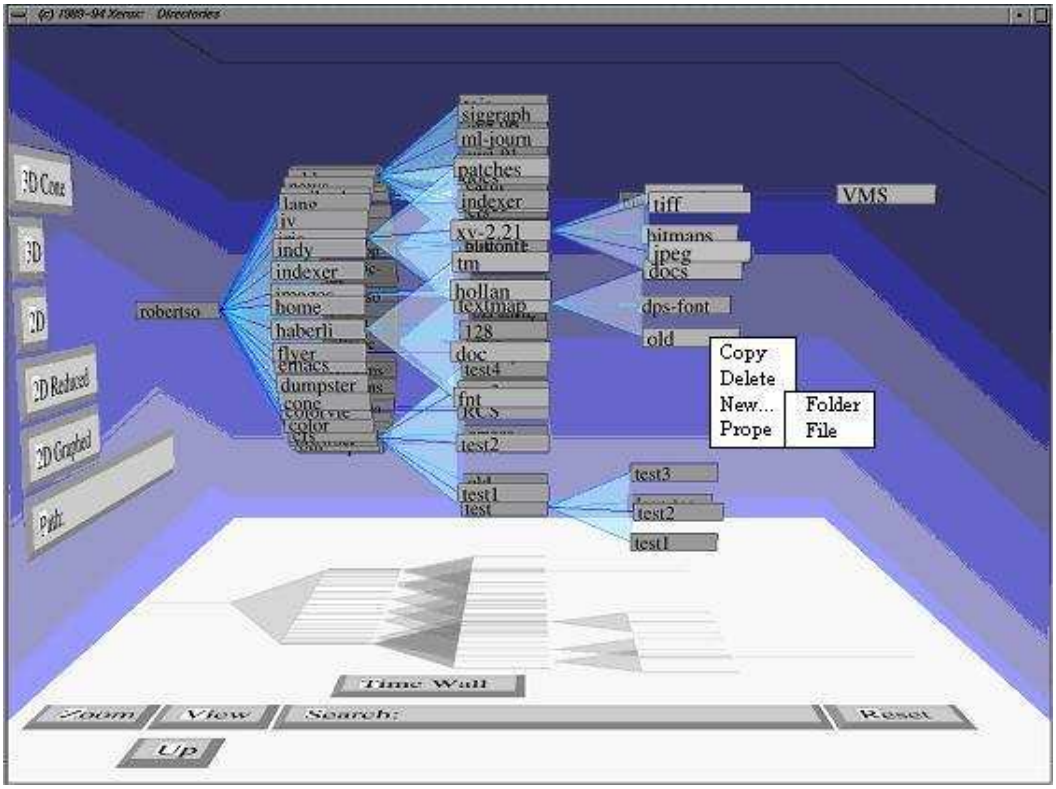
2.3.3.4.4. „Detalios informacijos“ pateikimo interfeiso prototipas (žr. 39 pav.)



39 pav. „Detalios informacijos“ interfeiso prototipas.

Ant norimo elemento paspaudus dešinį mygtuką ir pasirinkus funkciją „Properties“ parodoma detali informacija.

2.3.3.4.5. „Organizavimo“ sąsajos interfeiso prototipas (žr. 40 pav.)



40 pav. „Organizavimo“ interfeiso prototipas.

Pasirinkus norimą elementą ir paspaudus dešinį pelės mygtuką, galima kopijuoti, ištrinti ar sukurti naują elementą.

2.3.3.5. Sąsajos darnos ir standartizavimo reikalavimai

Visi interfeiso pranešimai apie klaidas, perspėjimai, informuojantys pranešimai turi būti tarpusavyje suderinti, neturi klaidinti naudotojo, turi padėti jam orientuotis susidariusioje situacijoje.

Interfeiso atitikimo standartams reikalavimų nėra.

2.3.3.6. Pranešimų formulavimo reikalavimai

Pranešimai turi būti trumpi, formuluojami vienu, dviem sakiniais. Kiekvienas pranešimas turi būti pateikiamas atskirame lange.

2.3.4. Funkciniai reikalavimai

2.3.4.1. Peržiūra

2.3.4.1.1. Pradiniai duomenys: Sistema yra išjungta.

2.3.4.1.2. Veiksmai: Naudotojas paleidžia sistemą.

2.3.4.1.3. Rezultatas: Atidaromas naršyklės langas ir pateikiamas failų sistemos vaizdas.

2.3.4.2. Priartinimas

2.3.4.2.1. Pradiniai duomenys: Naudotojas nori priartinti tam tikrą failų sistemos elementą.

2.3.4.2.2. Veiksmai: Pasirenkamas norimas elementas ir įvykdoma priartinimo funkcija.

2.3.4.2.3. Rezultatas: Priartinamas pasirinktas elementas.

2.3.4.3. Filtravimas

2.3.4.3.1. Pradiniai duomenys: Naudotojas nori atmesti jį nedominančius elementus.

2.3.4.3.2. Veiksmai: Įvedami nereikalingų elementų kriterijai.

2.3.4.3.3. Rezultatas: Kriterijus atitinkantys elementai pašalinami iš failų sistemos vaizdo.

2.3.4.4. Detalios informacijos gavimas

2.3.4.4.1. Pradiniai duomenys: Naudotojas nori pamatyti tam tikro elemento ar elementų grupės detaliąją informaciją.

2.3.4.4.2. Veiksmai: Pasirenkamas norimas elementas ar elementų grupė ir įvykdoma detaliosios informacijos pateikimo funkcija.

2.3.4.4.3. Rezultatas: Pateikiama elemento ar elementų grupės detalioji informacija.

2.3.4.5. Organizavimas

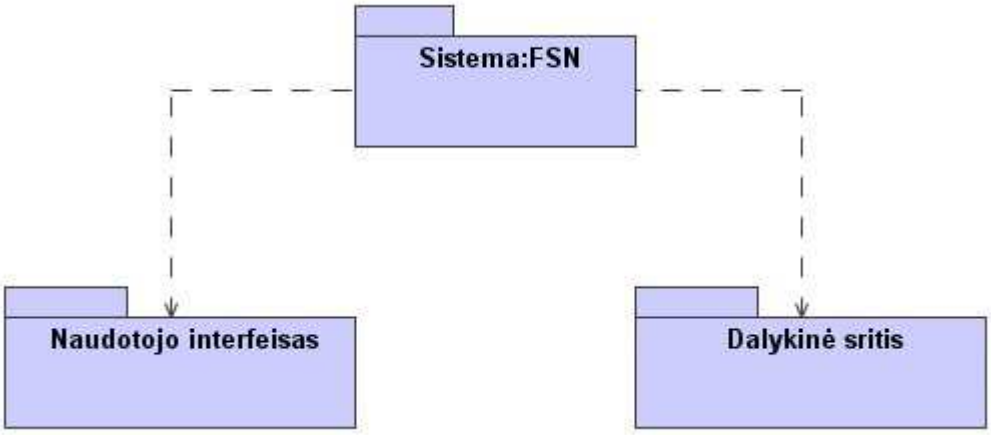
2.3.4.5.1. Pradiniai duomenys: Naudotojas nori rankiniu būdu sutvarkyti elementus, kad vėliau būtų lengviau juos rasti.

2.3.4.5.2. Veiksmai: Rankiniu būdų elementai perkeliami iš vieno katalogo į kitą, kopijuojami ar trinami.

2.3.4.5.3. Rezultatas: Pertvarkyta ar papildyta failų sistema.

2.3.5. Projektiniai reikalavimai

2.3.5.1. Sistemos dekompozicija



2.3.5.1.1. FSN: Reikalavimai sistemai

2.3.5.1.1.1.	[2.3.3.1.]	Dalykinė sistemos metafora
2.3.5.1.1.2.	[2.3.3.2.1.]	Peržiūra
2.3.5.1.1.3.	[2.3.3.2.2.]	Priartinimas
2.3.5.1.1.4.	[2.3.3.2.3.]	Filtravimas
2.3.5.1.1.5.	[2.3.3.2.4.]	Detali informacija
2.3.5.1.1.6.	[2.3.3.2.5.]	Organizavimas
2.3.5.1.1.7.	[2.3.3.3.]	Užduočių formulavimo kalba
2.3.5.1.1.8.	[2.3.3.4.]	Užduočių formulavimo būdas
2.3.5.1.1.9.	[2.3.3.5.]	Interfeiso darna ir standartizavimas
2.3.5.1.1.10.	[2.3.3.6.]	Pranešimų formulavimas
2.3.5.1.1.11.	[2.4.1.]	Peržiūra
2.3.5.1.1.12.	[2.4.2.]	Priartinimas
2.3.5.1.1.13.	[2.4.3.]	Filtravimas
2.3.5.1.1.14.	[2.4.4.]	Detali informacija
2.3.5.1.1.15.	[2.4.5.]	Organizavimas

2.3.5.1.2. FSN::Naudotojo interfeisas

Paketas yra logiškai vientisas, todėl nebeskaidomas.

Lokalizuoti sistemos reikalavimai šiam paketui nurodyti punkte: 3.3.5.2. Reikalavimų lokalizavimo matrica.

2.3.5.1.3. FSN::Dalykinė sritis

Paketas yra logiškai vientisas, todėl nebeskaidomas.

Lokalizuoti sistemos reikalavimai šiam paketui nurodyti punkte: 3.3.5.2 Reikalavimų lokalizavimo matrica.

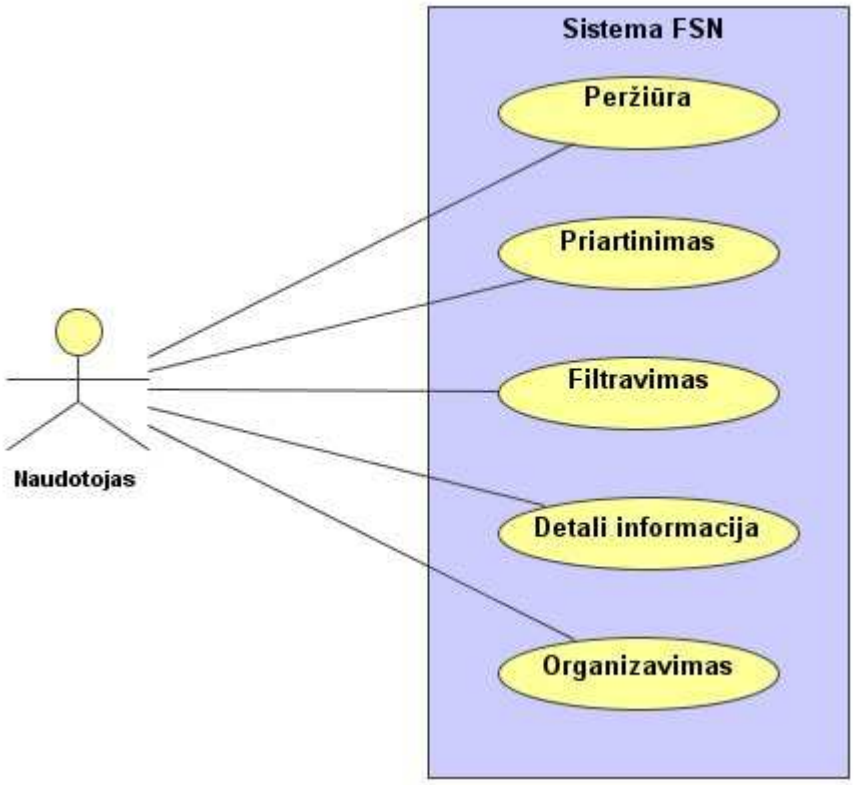
2.3.5.2. Reikalavimų lokalizavimo matrica

Reikalavimai \ Paketai		FSN	FSN::Vartotojo interfeisas	FSN::Dalykinė sritis
[2.3.3.1.]	Dalykinė sistemos metafora	+	+	
[2.3.3.2.1.]	Peržiūra	+	+	
[2.3.3.2.2.]	Priartinimas	+	+	
[2.3.3.2.3.]	Filtravimas	+	+	
[2.3.3.2.4.]	Detali informacija	+	+	
[2.3.3.2.5.]	Organizavimas	+	+	
[2.3.3.3.]	Užduočių formulavimo kalba	+	+	+
[2.3.3.4.]	Užduočių formulavimo būdas	+	+	+
[2.3.3.5.]	Interfeiso darna ir standartizavimas	+	+	+
[2.3.3.6.]	Pranešimų formulavimas	+	+	+
[2.4.1.]	Peržiūra	+		+
[2.4.2.]	Priartinimas	+		+
[2.4.3.]	Filtravimas	+		+
[2.4.4.]	Detali informacija	+		+
[2.4.5.]	Organizavimas	+		+

2.3.6. Sistemos architektūra

2.3.6.1. Užduotys ir jų vykdymo scenarijai

2.3.6.1.1. Sistemos vykdomos užduotys



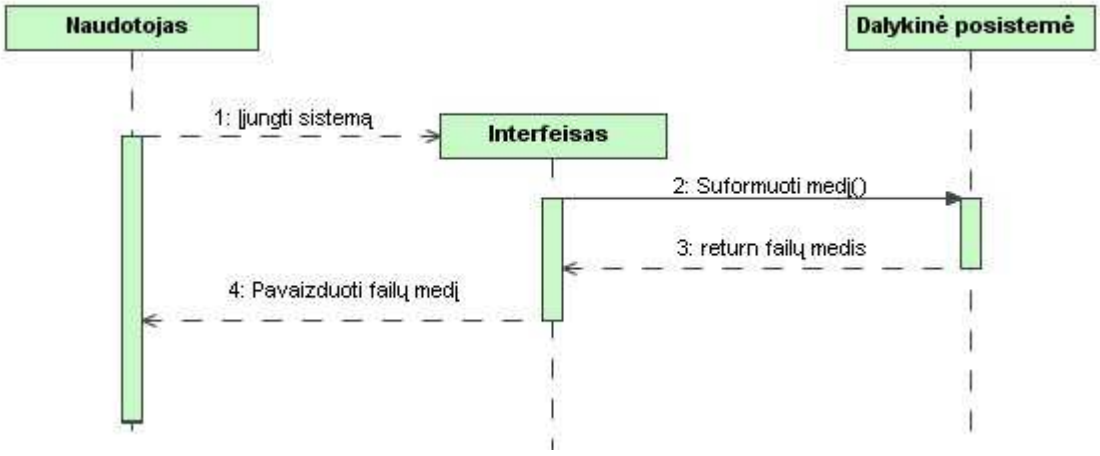
Užduočių įgyvendinimas:

Versija: 1.0

Sistema: FSN

Pirminis agentas: naudotojas

2.3.6.1.2. Užduoties „Peržiūra“ įgyvendinimas



Siekiamas tikslas: peržiūrėti failų medį

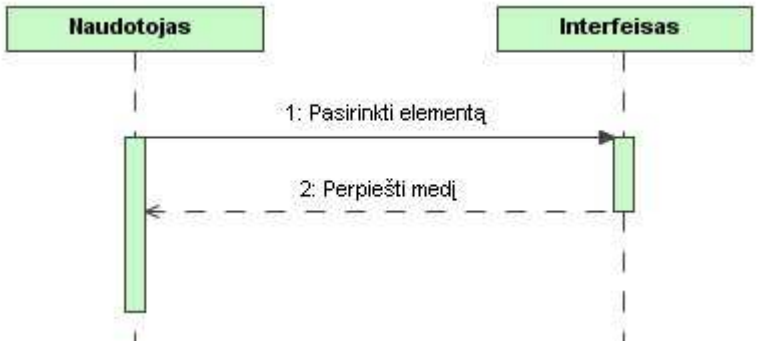
“Prieš” sąlygos: sistema neįjungta

“Po” sąlygos: sistema įjungta ir pavaizduotas failų medis

Scenarijus:

1. Naudotojas įjungia sistemą
2. Sistema suformuoja failų medį
3. Sistema pavaizduoja failų medį

2.3.6.1.3. Užduoties „Priartinimas“ įgyvendinimas



Siekiamas tikslas: priartinti elementą

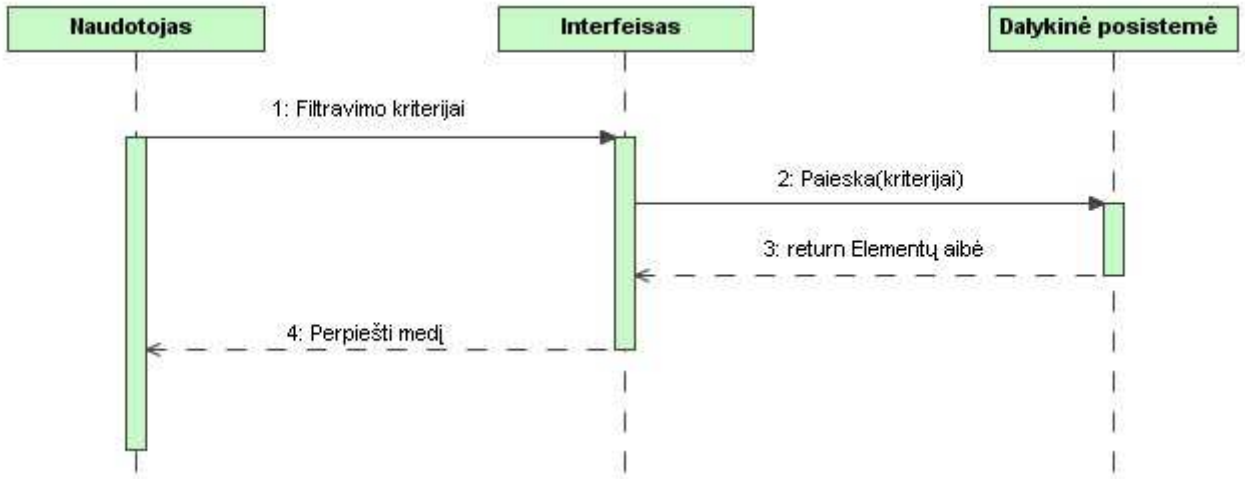
“Prieš” sąlygos: sistema įjungta

“Po” sąlygos: pavaizduotas failų medis taip, kad jo priekyje yra pasirinktas elementas ir kelias iki jo

Scenarijus:

1. Naudotojas pasirenka elementą
2. Sistema pavaizduoja atnaujintą failų medžio vaizdą

2.3.6.1.4. Užduoties „Filtravimas“ įgyvendinimas



Siekiamas tikslas: išskirti elementų poaibį

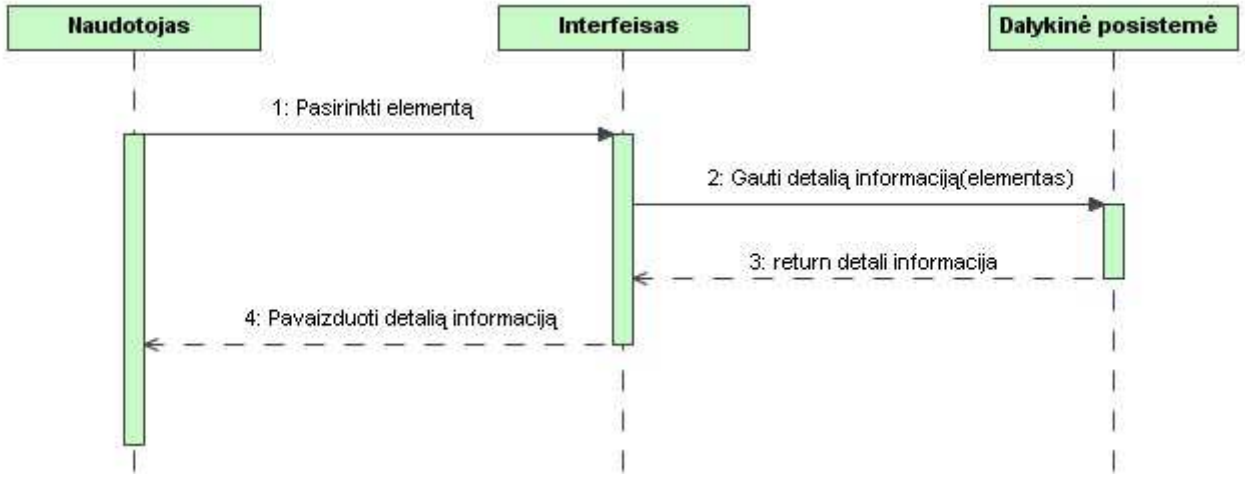
“Prieš” sąlygos: sistema įjungta

“Po” sąlygos: pavaizduotas kriterijus atitinkančių elementų medis

Scenarijus:

1. Naudotojas įveda paieškos kriterijus
2. Sistema suranda elementus, atitinkančius kriterijus
3. Sistema pavaizduoja rastų elementų failų medį

2.3.6.1.5. Užduoties „Detali informacija“ įgyvendinimas



Siekiamas tikslas: peržiūrėti detalią elemento informaciją

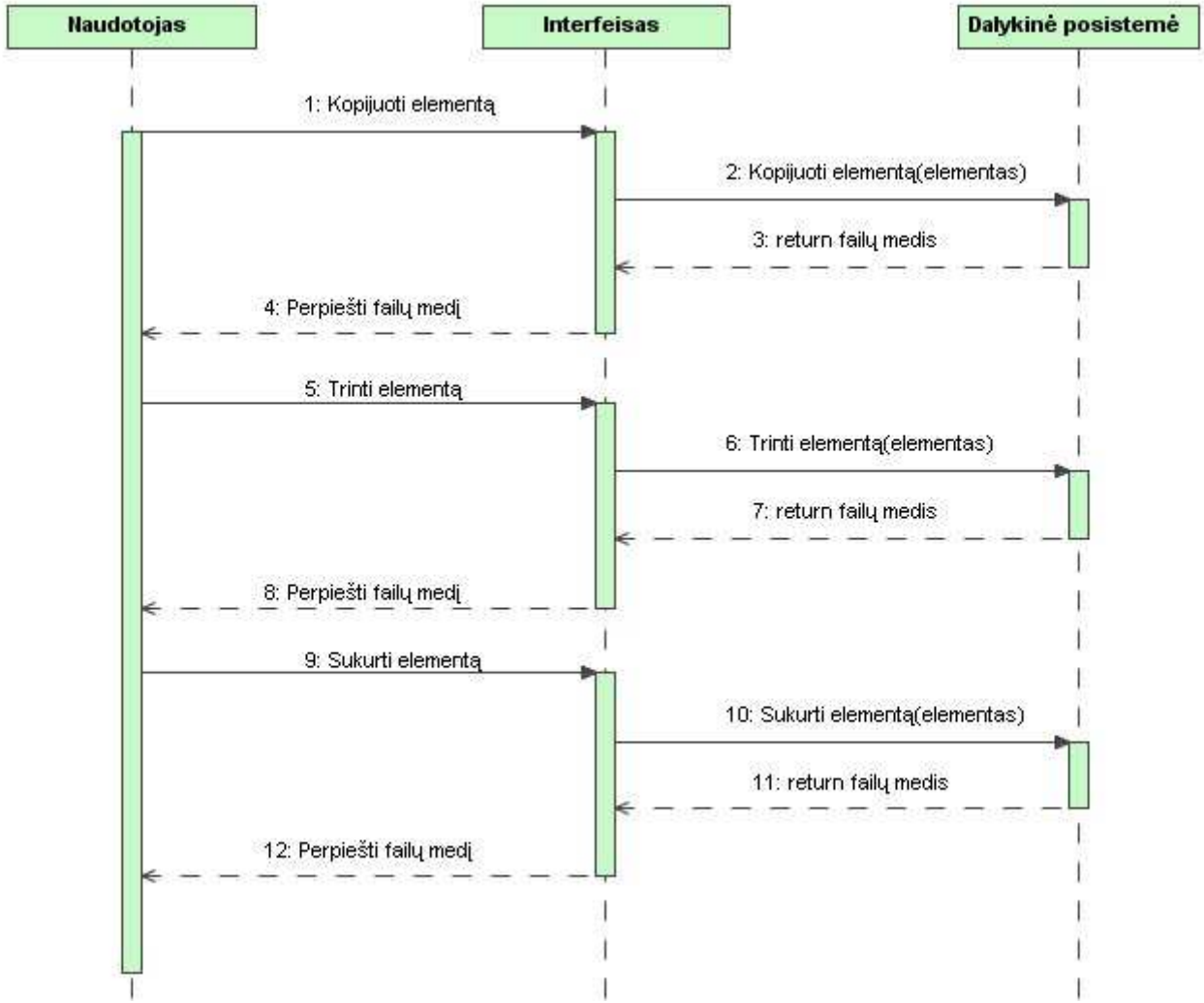
“Prieš” sąlygos: sistema įjungta

“Po” sąlygos: pateikta detali pasirinkto elemento informacija

Scenarijus:

1. Naudotojas pasirenka elementą ir funkciją detalios informacijos pateikimui
2. Sistema gauna detalią elemento informaciją
3. Sistema pateikia detalią pasirinkto elemento informaciją

2.3.6.1.6. Užduoties „Organizavimas“ įgyvendinimas



Siekiamas tikslas: tvarkyti failų sistemos elementus

“Prieš” sąlygos: sistema įjungta

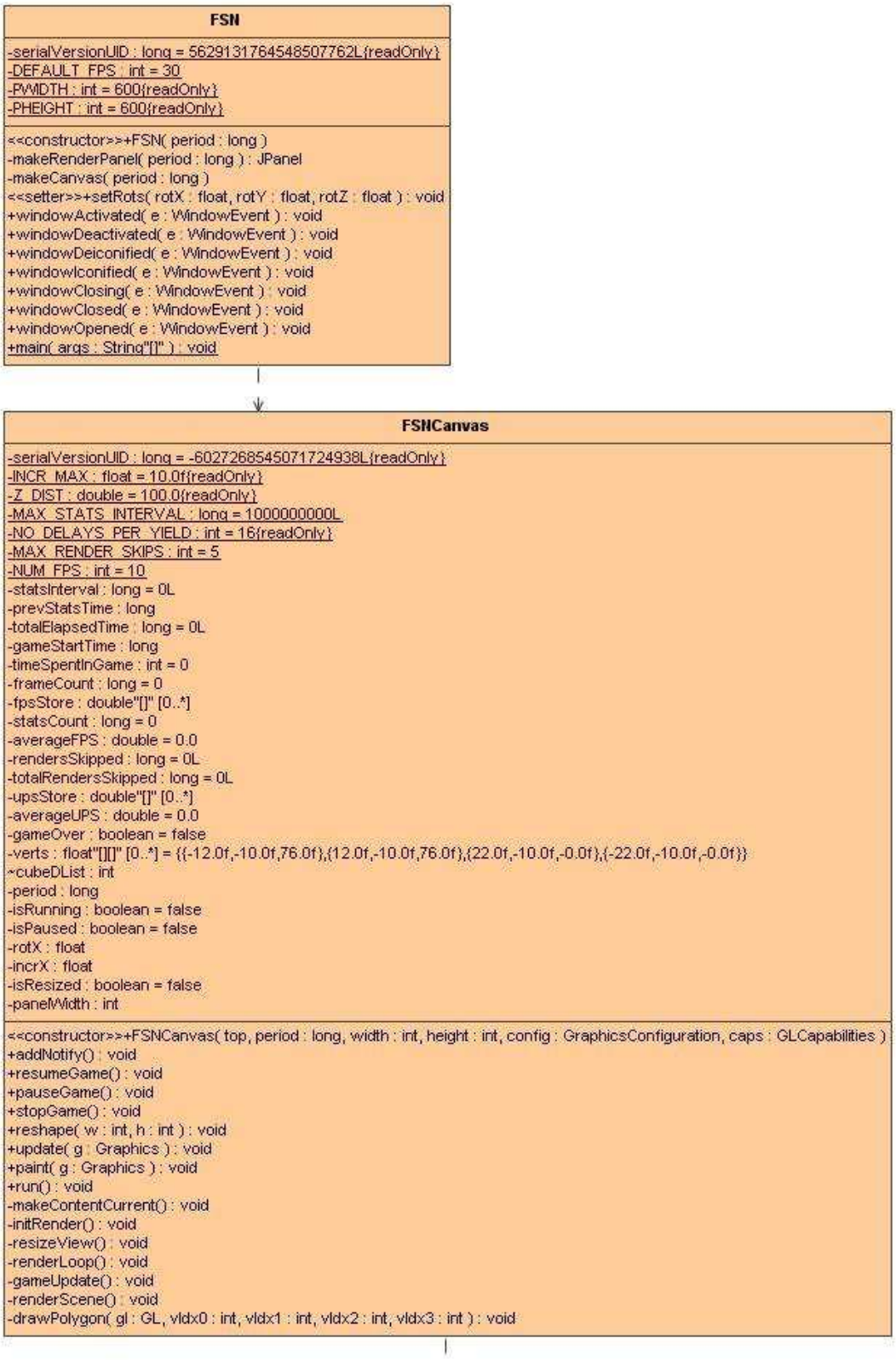
“Po” sąlygos: pakeistas ir pavaizduotas failų medis

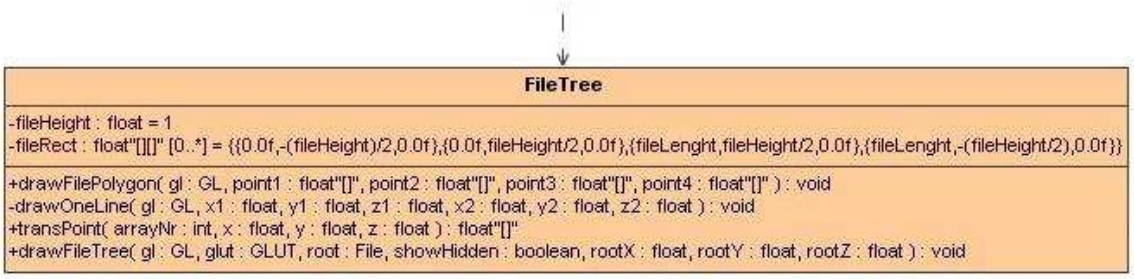
Scenarijus:

1. Naudotojas nukopijuoja elementą
2. Sistema nukopijuoja elementą
3. Sistema pavaizduoja atnaujintą failų medį
4. Naudotojas ištrina elementą
5. Sistema ištrina elementą

6. Sistema pavaizduoja atnaujintą failų medį
7. Naudotojas sukuria elementą
8. Sistema sukuria elementą
9. Sistema pavaizduoja atnaujintą failų medį

2.3.6.2. Struktūrinis programų sistemos modelis

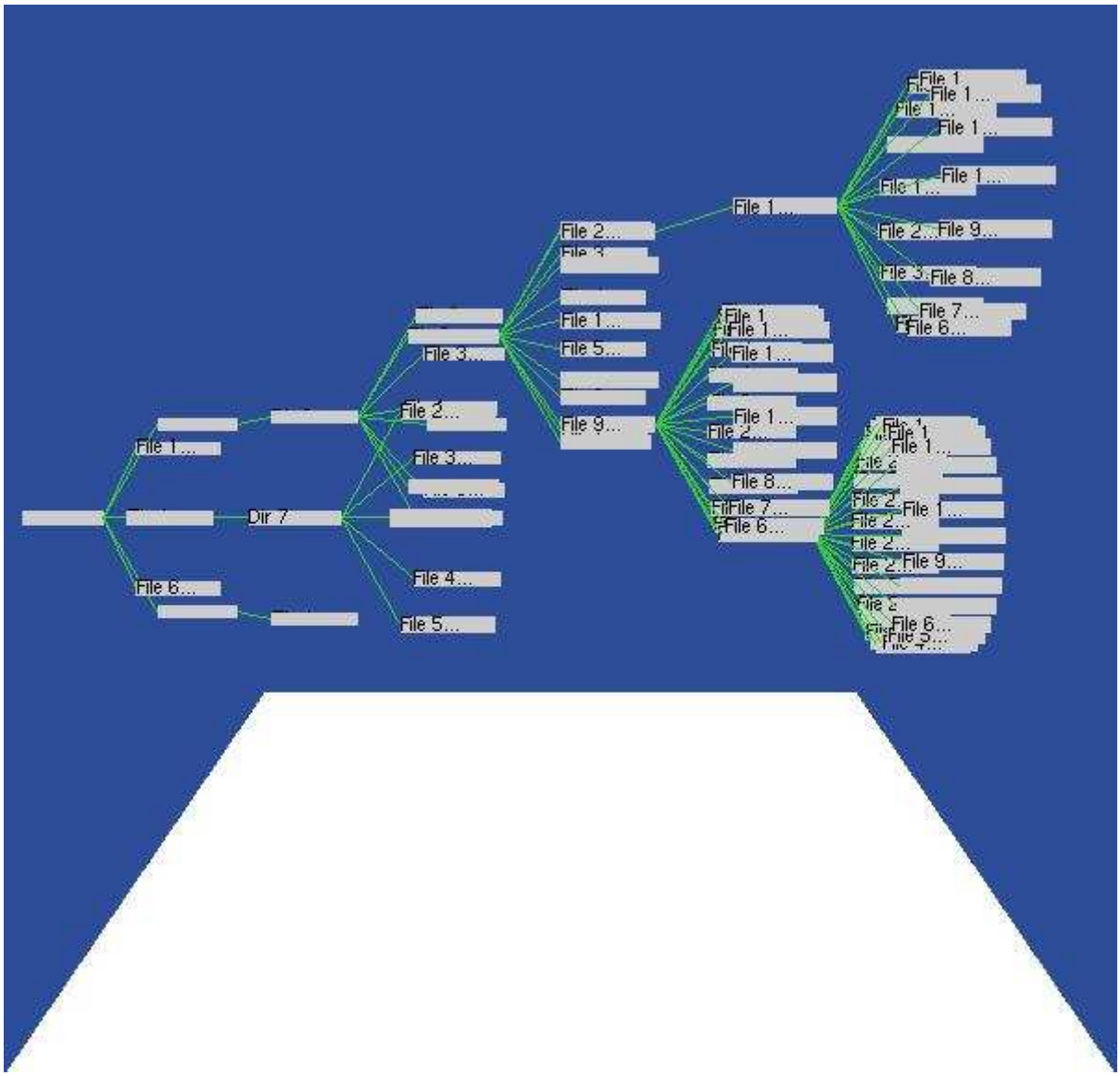




3. Failų naršyklės prototipas

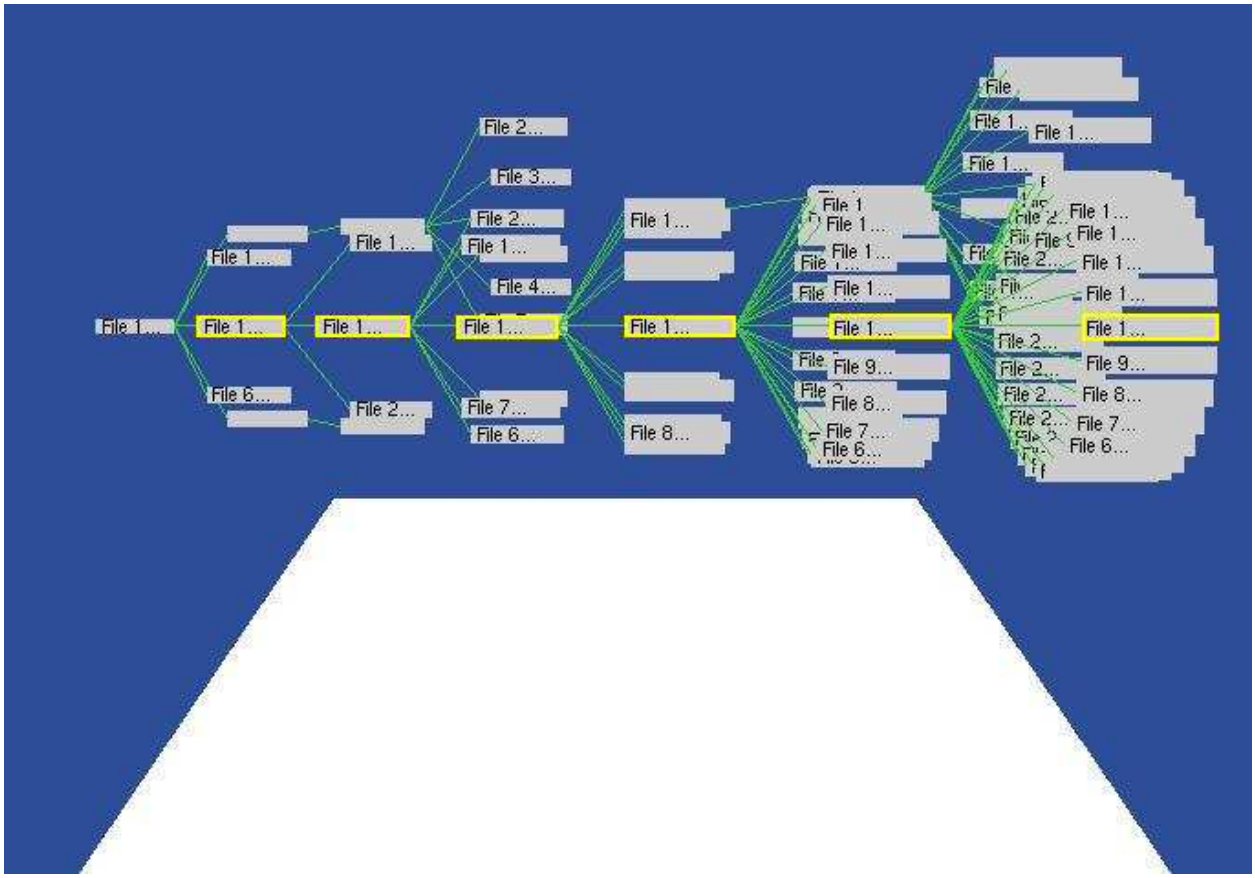
3.1. FSN interfeisų pavyzdžiai

Remiantis projektine dalimi buvo sukurta sistema FSN. Failų sistemos vaizdas pateikiamas kūginio medžio pavidalu. (žr. 41 pav.) Stačiakampio formos laukeliai vaizduoja failus arba katalogus. Atitinkamai pagal failų sistemos ryšius, jie susieti linijomis.



41 pav. NFS failų sistemos vaizdas.

Pasirinkus norimą elementą, yra išryškinamas jo kelias – elementai pavaizduojami kūginio medžio priešakinėje dalyje ir apvedami geltonai. (žr. 42 pav.)



42 pav. Išryškintas kelias iki pasirinkto elemento.

3.2. Problemos ir iššūkiai

Sistemos kūrimui buvo reikalingos OpenGL žinios, kadangi dirbant su JOGL technologija yra operuojama OpenGL klasėmis. Ši technologija man buvo nauja, todėl reikėjo su ja susipažinti. Suprasti jos veikimo ir naudojimo principus.

Elementų išdėstymas trimatėje erdvėje yra paremtas koordinačių skaičiavimais. FSN lange koordinačių sankirtos taškas yra jo centre. Šakninio elemento (katalogo) stačiakampis patalpinamas ant X ir Z ašių pradžios, koordinatėse x = 0, y = -40, z = 0. Šiame kataloge esantys elementai (vaikai) paslenkami Y ašimi į dešinę ir išdėliojami apskritimu aplink Z ašį. Elementai išdėstomi vienodais atstumais vienas nuo kito. Šiom koordinatėm apskaičiuoti naudotos išvestinės formulės iš formulės x2 + y2 = r2. Skaičiavimas pateikiamas pseudo kodu:

1. KampasTarpElementų = π * 2 / elementųSkaičius;
2. for (elementųSkaičius) start:
3. PilnasKampas = KampasTarpElementų * elementoNr;
4. y = tėvoY + sin(pilnasKampas) * r;

5. z = tėvoZ + cos(pilnasKampas) * r;

6. end;

Pirmoje eilutėje apskaičiuojamas kampas tarp elementų, išsidėsčiusių apskritimu. Tuomet vykdomas ciklas kiekvienam vaiko elementui. Trečioje eilutėje apskaičiuojamas kampas nuo 0 laipsnių, apskritimo pradžios. Išvestinėmis formulėmis apskaičiuojamos Y ir Z koordinatės. X koordinatė visiems elementams yra ta pati. Tai vykdo šis programinis kodas:

```
/** Metodas, piešiantis failų medį. */
public void drawFileTree(GL gl, GLUT glut, File root, boolean showHidden,
float rootX, float rootY, float rootZ)
{
//failų sistema
FileSystemView fsv = FileSystemView.getFileSystemView();

//duoto katalogo "root" turinys
File[] dirContent = fsv.getFiles(root, showHidden);

//apskaičiuojamas kampas
double radius = Math.PI * 2 / dirContent.length;
final float x = rootX + 4;
final float r = 7;


for (int i = 0; i < dirContent.length; i++)
{
//apskaičiuojamos koordinatės
File file = dirContent[i];
double fileRadius = radius * i;
final float z = new Float(rootZ + Math.cos(fileRadius) * r).floatValue();
final float y = new Float(rootY + Math.sin(fileRadius) * r).floatValue();

//piešiama linija
gl.glColor3f(0.2f, 0.8f, 0.2f);
drawOneLine(gl, rootX, rootY, rootZ, x, y, z);

   <...>
}
```

Tai atliekant rekursyviai su kiekvienu katalogu gaunamas kūgio pavidalo medis 3D erdvėje. Ta pati formulė buvo naudojama ir elemento pavadinimo užrašymui ant stačiakampio. Tai atlieka šis programinis kodas:

```
String fileName = file.getName();

if (fileName.length() > 8)
    fileName = " " + fileName.substring(0, 6) + "...";

//užrašomas pavadinimas
gl.glColor3f(0.0f, 0.0f, 0.0f);
gl.glRasterPos3f(x, y - 0.37f, z);
glut.glutBitmapString(GLUT.BITMAP_HELVETICA_10, fileName);

//piešiamas stačiakampis
gl.glColor3f(0.8f, 0.8f, 0.8f);
```

```
drawFilePolygon(gl, transPoint(0, x, y, z), transPoint(1, x, y, z),
transPoint(2, x, y, z), transPoint(3,
                          x, y, z));

//jei tai katalogas, kviečiam šią funkciją rekursyviai
if (file.isDirectory())
{
    drawFileTree(gl, glut, file, false, x + fileLenght, y, z);
}
```

Pasirinkto elemento kelio išryškinimui reikia visus elementus, įeinančius į tą kelią patalpinti medžio priešakinėje dalyje. Perpiešiant failų medį, kiekviename lygyje elementas, įeinantis į kelią piešiamas taip, kad jo z koordinatė būtų didžiausia, lyginant su kitais to pačio kūgio elementais. Visi programiniai kodai pateikti priede. (žr. 7.1. Programinis kodas)

3.3. Tolimesnio darbo siūlymai

Kadangi FSN yra prototipas, tai šio darbo tikslas nebuvo sukurti pilną funkcionalumą turinčią sistemą. Įgyvendintos svarbiausios funkcijos susijusios su pavaizdavimu. Tačiau nerealizuota failų paieška, jų organizavimas: trynimo, kopijavimo, sukūrimo, detalios informacijos pateikimo funkcijos.

Sekančiame etape siūloma šias funkcijas realizuoti. Interfeise jau esamiems failų stačiakampiams priskirti JoglEventListener klasės objektą, skirtą reaguoti į pelės ar klaviatūros veiksmus, o dalykinėje srityje pasinaudoti programavimo kalbos Java teikiamomis failų valdymo galimybėmis – klasės File metodais: createNewFile() – naujo failo sukūrimui, delete() – failo ar katalogo ištrynimui, mkDir() – katalogo sukūrimui. Taip pat siūloma sukurti sistemos instaliaciją naudojant instaliacijos vedlį.

3.4. Apibendrinimas

Sukurtas FSN sistemos prototipas tenkina iškeltus reikalavimus. Visas funkcionalumas nėra realizuotas – nesukurtos failų tvarkymo funkcijos, tačiau failų medis nuo duoto šakninio katalogo pavaizduojamas 3D kūginiu medžiu. Sistema be trukdžių pavaizduoja ir didesnius nei 1000 failų medžius. Visas medis yra matomas programos lange.

Pasirinktos technologijos JOGL galimybių užteko tiek jau sukurtam prototipui, tiek ir nesukurtoms funkcijos. Esant ~3000 failų medžiui, sistemos įjungimo metu jis nupaišomas per 1-2 sekundes. Sistemos veikimo metus iš naujo perpiešiant medį kas 10 milisekundžių trukdžiai nepastebimi.

Išvados

Šiame darbe buvo atlikta teorinė trimačių interfeisų vizualizavimo analizė. Kadangi failų tvarkymo sistema yra hierarchinė, tai apžvelgėme tik hierarchinių struktūrų pavaizdavimo būdus. Vizualizavimo būdui buvo iškelti 3 kriterijai:

1. Vykdyti šias funkcijas: peržiūra, priartinimas, ryšiai.
2. Vaizduoti didelės apimties medžius, sudarytus iš >1000 failų bendrai visuose lygiuose. Failų medis turi tilpti matomo programos lango ribose.
3. Katalogam ir failam pateikti papildomą statistinę informaciją.

Atsižvelgiant į juos buvo pasirinktas kūginio medžio būdas. Realizavus sistemos FSN prototipą pastebėta, jog šis vizualizavimo būdas pasirinktas teisingai ir yra tinkamas dideliems failų medžiams vaizduoti. Esant ~3000 failų dydžio medžiui, jis visas yra aiškiai matomas ir telpa programos lango ribose.

Buvo atlikta sukurtų 3D failų naršyklių analizė, kurios metu kiekviena iš 10 sistemų buvo vertinama pagal vizualizacijos būdui pasirinktus kriterijus. Nei viena sistema ir nei vienas panaudotas vizualizacijos būdas netenkina kuriamai sistemai iškeltų kriterijų. Nei viena sistema nenaudojo kūginio medžio vizualizacijos. Todėl kuriamas sistemos prototipas FSN sėkmingai pritaikė dar nerealizuotą failų medžio vaizdavimo būdą.

Kitoje dalyje pasirinkta sistemos realizavimo technologija. Jai iškelti 3 kriterijai:

1. Greitas sistemos atsakas, be vizualiai pastebimų trukdžių.
2. Kuo aukštesnis lygio programavimas.
3. Pastovus technologijos atnaujinimas.

Buvo palygintos dvi technologijos – Java 3D ir JOGL. Sistemos realizacijai buvo pasirinkta visus kriterijus tenkinusi, atsako greičiu pranašesnė ir automatiškai atnaujinama technologija JOGL. Esant ~3000 failų dydžio medžiui sistema FSN be pastebimų trukdžių failų medį perpaišo kas 10 milisekundžių.

Buvo aprašyta kuriama 3D naudotojo interfeiso failų sistemos naršyklė „FSN“. Apibrėžti „FSN“ sistemos reikalavimai, pateikti interfeiso prototipai. Remiantis šiais reikalavimais sukurtas sistemos prototipas. Sistemos FSN interfeisas atitiko pateiktus interfeiso prototipus. Kuriant sistemą didžiausios problemos kilo atliekant matematinius veiksmus bei įsisavinant OpenGL technologiją. Nebuvo realizuotos failų tvarkymo funkcijos, tačiau pateikti tikslūs siūlymai, kaip tai atlikti.

Literatūros sąrašas

[RCL+98] George G. Robertson, Mary Czerwinski, Kavin Larson, Daniel C. Robbins, David Thiel, Maarten van Dantzich. Data Mountain: Using Spatial Memory for Document Management. Microsoft Resarch, WA, USA. 1998. http://www.microsoft.com/usability/UEPostings/p153-robertson.pdf

[CM+00] Andy Cockburny, Bruce McKenzie. An Evaluation of Cone Trees. Department of Comtuper Sciense, University of Canterbury, Christchurch, New Zealand. 2000. http://www.cosc.canterbury.ac.nz/andrew.cockburn/papers/conesBCSHCI.pdf

[RMC91] George G. Robertson, Jock D. Mackinlay, Stuart K. Card. Cone Trees: Animated 3D visualizations of Hierarchical Information. Xerox Palo Alto Resarch Center, Palo Alto, CA, USA. 1991. http://www.cs.auckland.ac.nz/courses/compsci705s1c/assignments/seminarreports/pyap011%20Seminar_Final_Report_pyap011.pdf

[MHG06] Daisuke Mizukoshi, Yukio Hori, Tomonori Gotoh. Extension models of Cone Tree Visualizations to Large scale Knowledge base with Semantic Relations. Department of Information Science, Kanagawa University, Kanagawa, Japan. 2006. http://wscg.zcu.cz/WSCG2006/Papers_2006/Poster/C89-full.pdf

[AGL+03] Frank Althoff, Gregor Mc Glaun, Manfred Lang, Gerhard Rigoll. Comparing an Innovative 3D and a Standart 2D User Interface for Automative Infotainment Applications. Lehrstuhl für Mensch-Maschine-Kommunikation, Technische Universität München. 2003. http://wotan.liu.edu/docis/dbl/mcmcmc/2003___CAI3AA.htm

[And02] Keith Andrews. Visualising Information Structures. Graz University of Technology, Styria, Austria. 2002.

[Nor05] Chris North. Information Visualization. Virginia Polytechnic Institute and Sate Unversity, Blacksburg, USA. 2005.

http://infovis.cs.vt.edu/papers/HHFE-infovis.pdf

[WW+99] Jake J van Wijk, Huub van de Wetering. Cushiom Treemaps: Visualization of Hierarchical Information. Eindhoven University of Technology, Eindhoven, The Netherlands. 1999. http://www.win.tue.nl/~vanwijk/ctm.pdf

[NPM+97] D. A. Nation, C. Plaisant, G. Marchionini, A. Komlodi. Visualizing websites using a hierarchical table of contents browser: WebTOC. In Proc. 3rd Conference on Human Factors and the Web, Denver, Colorado, USA (1997). ftp://ftp.cs.umd.edu/pub/hcil/Demos/WebTOC/Paper/WebTOC.html

[SC+01] A. Smith, D. Clark. Building a Customized Tree View. IBM developerWorks Tutorial. 2001 http://www.ibm.com/developerworks/training/.

[Dyk91] P. Dykstra. XDU. Army Research Laboratory. 1991

[JS+91] B. Johnson, B. Shneiderman. Tree-Maps: A Space-Filling Approach to the Visualization of Hierarchical Information Structures. In Proc. IEEE Visualization ’91, pages 284–291, San Diego, California, USA 1991. IEEE Computer Society. http://www.cs.umd.edu/hcil/treemaps/.

[Wat99] M. Wattenberg. Visualizing the Stock Market. In CHI 99 Extended Abstracts, pages 188–189, Pittsburgh, Philadelphia, USA 1999. ACM. http://www.smartmoney.com/marketmap/.

[SZ+00] J. Stasko, E. Zhang. Focus+Context Display and Navigation Techniques for Enhancing Radial, Space-Filling Hierarchy Visualizations. In Proc. IEEE InfoVis 2000, pages 57–65, Salt Lake City, Utah, USA 2000. IEEE Computer Society. http://www.cc.gatech.edu/gvu/ii/sunburst/.

[Dav07] Andrew Davison . Pro Java 6 3D Game Development. Java 3D, JOGL, JInput and JOAL. Pages 377-439, USA 2007. APRES.

Priedai

Programinis kodas:

FSN.java

```
package fsn;

import java.awt.BorderLayout;
import java.awt.Container;
import java.awt.Dimension;
import java.awt.GraphicsConfiguration;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.ComponentAdapter;
import java.awt.event.ComponentEvent;
import java.awt.event.WindowEvent;
import java.awt.event.WindowListener;
import java.text.DecimalFormat;

import javax.media.opengl.AWTGraphicsConfiguration;
import javax.media.opengl.AWTGraphicsDevice;
import javax.media.opengl.GLCapabilities;
import javax.media.opengl.GLDrawableFactory;
import javax.swing.ImageIcon;
import javax.swing.JFrame;
import javax.swing.JMenu;
import javax.swing.JMenuBar;
import javax.swing.JMenuItem;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
import javax.swing.JTextField;

public class FSN extends JFrame implements WindowListener
{
    private static final long serialVersionUID = 5629131764548507762L;

    private static int DEFAULT_FPS = 30;

    private static final int PWIDTH = 600;
    private static final int PHEIGHT = 600;

    private JMenuBar mBar = new JMenuBar();

    private JMenu menu = new JMenu("Meniu");
    private JMenu help = new JMenu("Pagalba");

    private ImageIcon closeIcon = new ImageIcon("Ikonos/iseiti.png");
    private ImageIcon aboutIcon = new ImageIcon("Ikonos/apie.png");

    private JMenuItem close = new JMenuItem("Išeiti", closeIcon);
    private JMenuItem about = new JMenuItem("Apie Murzių", aboutIcon);

    private FSNCanvas canvas;

    private JTextField rotsTF; // displays cube rotations
    private DecimalFormat df = new DecimalFormat("0.#"); // 1 dp

    ActionListener menuBarListener = new MenuBarListener();

    public FSN(long period)
    {
```

```
            super("FSN (Active)");

            Container c = getContentPane();
            c.setLayout(new BorderLayout());

            menu.add(close);
            help.add(about);

            mBar.add(menu);
            mBar.add(help);

            JPanel renderPane = makeRenderPanel(period);
            renderPane.add(mBar, BorderLayout.NORTH);

            c.add(renderPane, BorderLayout.CENTER);

            addWindowListener(this);
            close.addActionListener(menuBarListener);
            about.addActionListener(menuBarListener);

            pack();
            setVisible(true);
    }

    public class MenuBarListener implements ActionListener
    {
            public void actionPerformed(ActionEvent e)
            {
                JMenuItem kuris = (JMenuItem) e.getSource();

                if (kuris == close)
                {
                        System.exit(0);
                }

                if (kuris == about)
                {
                        JOptionPane.showMessageDialog(null, "FSN\n" + "Versija
1.0\n\n" + "Failų sistemos naršyklė.\n"
                                + "©2005 Gediminas Mitrikevičius", "Apie
Murzių", JOptionPane.INFORMATION_MESSAGE);
                }
            }
    }

    private JPanel makeRenderPanel(long period)
    {
            JPanel renderPane = new JPanel();
            renderPane.setLayout(new BorderLayout());
            renderPane.setOpaque(false);
            renderPane.setPreferredSize(new Dimension(PWIDTH, PHEIGHT));

            canvas = makeCanvas(period);
            renderPane.add(canvas, BorderLayout.CENTER);

            canvas.setFocusable(true);
            canvas.requestFocus(); // the canvas now has focus, so receives key
            // events

            // detect window resizes, and reshape the canvas accordingly
            renderPane.addComponentListener(new ComponentAdapter()
            {
                public void componentResized(ComponentEvent evt)
                {
```

```
                    Dimension d = evt.getComponent().getSize();
                    // System.out.println("New size: " + d);
                    canvas.reshape(d.width, d.height);
                } // end of componentResized()
            });

            return renderPane;
    }

    private FSNCanvas makeCanvas(long period)
    {
            // get a configuration suitable for an AWT Canvas (for FSNCanvas)
            GLCapabilities caps = new GLCapabilities();

            AWTGraphicsDevice dev = new AWTGraphicsDevice(null);
            AWTGraphicsConfiguration awtConfig = (AWTGraphicsConfiguration)
GLDrawableFactory.getFactory()
                    .chooseGraphicsConfiguration(caps, null, dev);

            GraphicsConfiguration config = null;
            if (awtConfig != null)
                config = awtConfig.getGraphicsConfiguration();

            return new FSNCanvas(this, period, PWIDTH, PHEIGHT, config, caps);
    }

    public void setRots(float rotX, float rotY, float rotZ)
    // called from FSNCanvas to show cube rotations
    {
            rotsTF.setText("Rotations: (" + df.format(rotX) + ", " +
df.format(rotY) + ", " + df.format(rotZ) + ")");
    }

    public void windowActivated(WindowEvent e)
    {
            canvas.resumeGame();
    }

    public void windowDeactivated(WindowEvent e)
    {
            canvas.pauseGame();
    }

    public void windowDeiconified(WindowEvent e)
    {
            canvas.resumeGame();
    }

    public void windowIconified(WindowEvent e)
    {
            canvas.pauseGame();
    }

    public void windowClosing(WindowEvent e)
    {
            canvas.stopGame();
    }

    public void windowClosed(WindowEvent e)
    {
    }

    public void windowOpened(WindowEvent e)
    {
```

```
    }

    public static void main(String[] args)
    {
           int fps = DEFAULT_FPS;
           if (args.length != 0)
               fps = Integer.parseInt(args[0]);

           long period = (long) 1000.0 / fps;
           System.out.println("fps: " + fps + "; period: " + period + " ms");

           new FSN(period * 1000000L);
    }
}
```

FSNCanvas.java

```
package fsn;

import java.awt.Canvas;
import java.awt.Graphics;
import java.awt.GraphicsConfiguration;
import java.io.File;

import javax.media.opengl.GL;
import javax.media.opengl.GLCapabilities;
import javax.media.opengl.GLContext;
import javax.media.opengl.GLDrawable;
import javax.media.opengl.GLDrawableFactory;
import javax.media.opengl.glu.GLU;

import com.sun.opengl.util.GLUT;

public class FSNCanvas extends Canvas implements Runnable
{
    private static final long serialVersionUID = -6027268545071724938L;

    private static final float INCR_MAX = 10.0f; // for rotation increments
    private static final double Z_DIST = 100.0; // for the camera position

    // statistic constants
    private static long MAX_STATS_INTERVAL = 1000000000L;
    // record stats every 1 second (roughly)

    private static final int NO_DELAYS_PER_YIELD = 16;
    /*
     * Number of iterations with a sleep delay of 0 ms before the animation
     * thread yields to other running threads.
     */

    private static int MAX_RENDER_SKIPS = 5; // was 2;
    // no. of renders that can be skipped in any one animation loop
    // i.e the games state is updated but not rendered

    private static int NUM_FPS = 10;
    // number of FPS values stored to get an average

    // used for gathering statistics
    private long statsInterval = 0L; // in ns
    private long prevStatsTime;
    private long totalElapsedTime = 0L;
    private long gameStartTime;
    private int timeSpentInGame = 0; // in seconds
```

```
    private long frameCount = 0;
    private double fpsStore[];
    private long statsCount = 0;
    private double averageFPS = 0.0;

    private long rendersSkipped = 0L;
    private long totalRendersSkipped = 0L;
    private double upsStore[];
    private double averageUPS = 0.0;

    // used at game termination
    private volatile boolean gameOver = false;

    //private String rootPath = "D:\\Mano_dokumentai\\Mokslai";
    private String rootPath = "D:\\Dir 1";

    // vertices for a cube of sides 2 units, centered on (0,0,0)
    /*
     * 2----3 | | 1----4
     */
    private float[][] verts =
    {
    { -12.0f, -13.0f, 76.0f },
    { 12.0f, -13.0f, 76.0f },
    { 22.0f, -13.0f, -0.0f },
    { -22.0f, -13.0f, -0.0f } };

    int cubeDList; // display list for displaying the cube

    private FSN top; // reference back to top-level JFrame

    private long period; // period between drawing in _nanosecs_

    private Thread animator; // the thread that performs the animation
    private volatile boolean isRunning = false; // used to stop the animation
    // thread
    private volatile boolean isPaused = false;

    // OpenGL
    private GLDrawable drawable; // the rendering 'surface'
    private GLContext context; // the rendering context (holds rendering
state
    // info)
    private GL gl;
    private GLU glu;
    private GLUT glut;

    // rotation variables
    private float rotX, rotY, rotZ; // total rotations in x,y,z axes
    private float incrX, incrY, incrZ; // increments for x,y,z rotations

    // window sizing
    private boolean isResized = false;
    private int panelWidth, panelHeight;

    public FSNCanvas(FSN top, long period, int width, int height,
GraphicsConfiguration config, GLCapabilities caps)
    {
          super(config);

          this.top = top;
          this.period = period;
          panelWidth = width;
```

```
            panelHeight = height;

            // get a rendering surface and a context for this canvas
            drawable = GLDrawableFactory.getFactory().getGLDrawable(this, caps,
null);
            context = drawable.createContext(null);

            // statistics initialization
            fpsStore = new double[NUM_FPS];
            upsStore = new double[NUM_FPS];
            for (int i = 0; i < NUM_FPS; i++)
            {
                fpsStore[i] = 0.0;
                upsStore[i] = 0.0;
            }
    } // end of FSNCanvas()

    public void addNotify()
    // wait for the canvas to be added to the JPanel before starting
    {
            super.addNotify(); // make the component displayable
            drawable.setRealized(true); // the canvas can now be rendering into

            // initialise and start the animation thread
            if (animator == null || !isRunning)
            {
                animator = new Thread(this);
                animator.start();
            }
    } // end of addNotify()

    // ------------- game life cycle methods ------------
    // called by the JFrame's window listener methods

    public void resumeGame()
    {
            isPaused = false;
    }

    public void pauseGame()
    {
            isPaused = true;
    }

    public void stopGame()
    {
            isRunning = false;
    }

    // ------------------------------------------------

    public void reshape(int w, int h)
    {
            isResized = true;
            if (h == 0)
                h = 1; // to avoid division by 0 in aspect ratio in
resizeView()
            panelWidth = w;
            panelHeight = h;
    } // end of reshape()

    public void update(Graphics g)
    {
    }
```

```
    public void paint(Graphics g)
    {
    }

    public void run()
    {
           // makeContentCurrent();

           initRender();
           renderLoop();

           // discard the rendering context and exit
           // context.release();
           context.destroy();
           System.exit(0);
    }

    private void makeContentCurrent()
    {
           try
           {
               while (context.makeCurrent() == GLContext.CONTEXT_NOT_CURRENT)
               {
                      System.out.println("Context not yet current...");
                      Thread.sleep(100);
               }
           } catch (InterruptedException e)
           {
               e.printStackTrace();
           }
    }

    private void initRender()
    {
           makeContentCurrent();

           gl = context.getGL();
           glu = new GLU();
           glut = new GLUT();

           resizeView();

           gl.glClearColor(0.18f, 0.3f, 0.6f, 0.0f);

           // z- (depth) buffer initialization for hidden surface removal
           gl.glEnable(GL.GL_DEPTH_TEST);

           // create a display list for drawing the cube
           cubeDList = gl.glGenLists(1);
           gl.glNewList(cubeDList, GL.GL_COMPILE);
           // drawColourCube(gl);
           gl.glEndList();

           context.release();
    }

    private void resizeView()
    {
           gl.glViewport(0, 0, panelWidth, panelHeight); // size of drawing
area

           gl.glMatrixMode(GL.GL_PROJECTION);
           gl.glLoadIdentity();
```

```
            glu.gluPerspective(45.0, (float) panelWidth / (float) panelHeight,
1, 100); // 5,
            // 100);
            // fov, aspect ratio, near & far clipping planes
    }

    private void renderLoop()
    {
            // timing-related variables
            long beforeTime, afterTime, timeDiff, sleepTime;
            long overSleepTime = 0L;
            int noDelays = 0;
            long excess = 0L;

            gameStartTime = System.nanoTime(); // J3DTimer.getValue();
            prevStatsTime = gameStartTime;
            beforeTime = gameStartTime;

            isRunning = true;

            while (isRunning)
            {
                makeContentCurrent();
                // gameUpdate();

                renderScene(); // rendering
                drawable.swapBuffers(); // put the scene onto the canvas
                // swap front and back buffers, making the new rendering
visible

                try
                {
                       Thread.sleep(1000L); // nano -> ms
                } catch (InterruptedException ex)
                {
                }

                afterTime = System.nanoTime();
                timeDiff = afterTime - beforeTime;
                sleepTime = (period - timeDiff) - overSleepTime;

                if (sleepTime > 0)
                { // some time left in this cycle
                       try
                       {
                           Thread.sleep(sleepTime / 1000000L); // nano -> ms
                       } catch (InterruptedException ex)
                       {
                       }
                       overSleepTime = (System.nanoTime() - afterTime) -
sleepTime;
                } else
                { // sleepTime <= 0; this cycle took longer than the period
                       excess -= sleepTime; // store excess time value
                       overSleepTime = 0L;

                       if (++noDelays >= NO_DELAYS_PER_YIELD)
                       {
                           Thread.yield();
                           noDelays = 0;
                       }
                }

                beforeTime = System.nanoTime(); // J3DTimer.getValue();
```

```
                int skips = 0;
                while ((excess > period) && (skips < MAX_RENDER_SKIPS))
                {
                        excess -= period;
                        gameUpdate(); // update state but don't render
                        skips++;
                }
                rendersSkipped += skips;

                context.release();
                System.out.println("Running...");
            }
        }

        private void renderScene()
        {
            if (GLContext.getCurrent() == null)
            {
                System.out.println("Current context is null");
                System.exit(0);
            }

            if (isResized)
            { // resize the drawable if necessary
                resizeView();
                isResized = false;
            }

            // clear colour and depth buffers
            gl.glClear(GL.GL_COLOR_BUFFER_BIT | GL.GL_DEPTH_BUFFER_BIT);

            gl.glMatrixMode(GL.GL_MODELVIEW);
            gl.glLoadIdentity();

            glu.gluLookAt(0, 0, Z_DIST, 0, 0, 0, 0, 1, 0); // position camera

            gl.glColor3f(1.0f, 1.0f, 1.0f);
            drawPolygon(gl, 0, 1, 2, 3); // žemės plokštuma

            FileTree fileTree = new FileTree();

            gl.glColor3f(0.8f, 0.8f, 0.8f);
            fileTree.drawFilePolygon(gl, fileTree.transPoint(0, -40f, 0f, 0f),
fileTree.transPoint(1, -40f, 0f, 0f),
                    fileTree.transPoint(2, -40f, 0f, 0f),
fileTree.transPoint(3, -40f, 0f, 0f));

            fileTree.drawFileTree(gl, glut, new File(rootPath), false, -34f,
0f, 0f);

            if (gameOver)
                System.out.println("Game Over");
        }

        private void drawPolygon(GL gl, int vIdx0, int vIdx1, int vIdx2, int
vIdx3)
        {
            gl.glBegin(GL.GL_POLYGON);
            gl.glVertex3f(verts[vIdx0][0], verts[vIdx0][1], verts[vIdx0][2]);
            gl.glVertex3f(verts[vIdx1][0], verts[vIdx1][1], verts[vIdx1][2]);
            gl.glVertex3f(verts[vIdx2][0], verts[vIdx2][1], verts[vIdx2][2]);
            gl.glVertex3f(verts[vIdx3][0], verts[vIdx3][1], verts[vIdx3][2]);
            gl.glEnd();
```

```
    }

}
```

FileTree.java

```
package fsn;

import java.io.File;

import javax.media.opengl.GL;
import javax.swing.filechooser.FileSystemView;

import com.sun.opengl.util.GLUT;

public class FileTree
{
    private final float fileHeight = 1.2f, fileLenght = 6;

    private final float[][] fileRect =
    {
    { 0.0f, -(fileHeight) / 2, 0.0f },
    { 0.0f, fileHeight / 2, 0.0f },
    { fileLenght, fileHeight / 2, 0.0f },
    { fileLenght, -(fileHeight / 2), 0.0f } };

    public void drawFilePolygon(GL gl, float[] point1, float[] point2,
float[] point3, float[] point4)
    {
           gl.glBegin(GL.GL_POLYGON);
           gl.glVertex3f(point1[0], point1[1], point1[2]);
           gl.glVertex3f(point2[0], point2[1], point2[2]);
           gl.glVertex3f(point3[0], point3[1], point3[2]);
           gl.glVertex3f(point4[0], point4[1], point4[2]);
           gl.glEnd();
    }

    private void drawOneLine(GL gl, float x1, float y1, float z1, float x2,
float y2, float z2)
    {
           gl.glBegin(GL.GL_LINES);
           gl.glVertex3f((x1), (y1), (z1));
           gl.glVertex3f((x2), (y2), (z2));
           gl.glEnd();
    }

    public float[] transPoint(int arrayNr, float x, float y, float z)
    {
           return new float[]
           { fileRect[arrayNr][0] + x, fileRect[arrayNr][1] + y,
fileRect[arrayNr][2] + z };
    }

    /** Metodas, piešiantis failų medį. */
    public void drawFileTree(GL gl, GLUT glut, File root, boolean showHidden,
float rootX, float rootY, float rootZ)
    {
           //failų sistema
           FileSystemView fsv = FileSystemView.getFileSystemView();

           // duoto katalogo "root" turinys
           File[] dirContent = fsv.getFiles(root, showHidden);
```

```
            //apskaičiuojamas kampas
            double radius = Math.PI * 2 / dirContent.length;
            final float x = rootX + 4;
            final float r = 7;

            for (int i = 0; i < dirContent.length; i++)
            {
                //apskaičiuojamos koordinatės
                File file = dirContent[i];
                double fileRadius = radius * i;
                final float z = new Float(rootZ + Math.cos(fileRadius) *
r).floatValue();
                final float y = new Float(rootY + Math.sin(fileRadius) *
r).floatValue();

                //piešiama linija
                gl.glColor3f(0.2f, 0.8f, 0.2f);
                drawOneLine(gl, rootX, rootY, rootZ, x, y, z);

                String fileName = file.getName();

                if (fileName.length() > 8)
                       fileName = " " + fileName.substring(0, 6) + "...";

                //užrašomas pavadinimas
                gl.glColor3f(0.0f, 0.0f, 0.0f);
                gl.glRasterPos3f(x, y - 0.37f, z);
                glut.glutBitmapString(GLUT.BITMAP_HELVETICA_10, fileName);

                //piešiamas stačiakampis
                gl.glColor3f(0.8f, 0.8f, 0.8f);
                drawFilePolygon(gl, transPoint(0, x, y, z), transPoint(1, x, y,
z), transPoint(2, x, y, z), transPoint(3,
                           x, y, z));

                //jei tai katalogas, kviečiam šią funkciją rekursyviai
                if (file.isDirectory())
                {
                       drawFileTree(gl, glut, file, false, x + fileLenght, y,
z);
                }
            }
    }
}
```