KAUNO TECHNOLOGIJOS UNIVERSITETAS
INFORMATIKOS FAKULTETAS
INFORMACIJOS SISTEMŲ KATEDRA

Agnė Černiauskaitė

APLINKOS APSAUGOS INSPEKCIJOS INFORMACINĖS SISTEMOS POSISTEMIŲ PROJEKTAVIMAS IR PROGRAMINĖ REALIZACIJA

Magistro darbas

Vadovė
doc. dr. L. Nemuraitė

KAUNAS, 2005

KAUNO TECHNOLOGIJOS UNIVERSITETAS
INFORMATIKOS FAKULTETAS
INFORMACIJOS SISTEMŲ KATEDRA

TVIRTINU
Katedros vedėjas
doc. dr. R. Butleris
2005 05 23

APLINKOS APSAUGOS INSPEKCIJOS INFORMACINĖS SISTEMOS POSISTEMIŲ PROJEKTAVIMAS IR PROGRAMINĖ REALIZACIJA

Informatikos mokslo magistro baigiamasis darbas

Kalbos konsultantė
Lietuvių kalbos katedros lektorė
dr. J. Mikelionienė
2005 05 23

Vadovė
doc. dr. L. Nemuraitė
2005 05 23

Recenzentas
doc. V. Kiauleikis
2005 05 23

Atliko
IFM-9/2 gr. stud.
A. Černiauskaitė
2005 05 23

KAUNAS, 2005

SUMMARY

Nowadays the complexity and size of business systems is growing steadily and, as practise shows, the situation is not going to change in the nearest future. This necessitates difficulties not only for lifecycle of programmes' development but for maintenance and expansibility also. That's why, solving them, requires a broader perspective that views the system as consisting of other systems.

This work looks what kind of methods can be applied in this context. The analysis of modern technologies (Java programming language) and modelling methods (functional decomposition and object-oriented system engineering) are presented. Suggested system decomposition approach is the synthesis of already existing ones.

In this work new method has been applied for the development of experimental Information System of Territorial Departments of Environmental Inspection. The prototype of the mentioned system has been realized by using J2EE (Java 2, Enterprise Edition) technologies. Developed software implements the requirements for usage and maintenance (large number of the users, security rights, compatibility, centralised administration, etc.) of large distributed systems. Suggested approach may be used while expanding the existing software or developing similar applications.

TURINYS

Paveikslų sąrašas

Lentelių sąrašas

ĮVADAS

Temos aktualumas. Vis didėjanti šiandienos sistemų apimtis bei augantis jų sudėtingumas verčia galvoti apie sudėtingų sistemų, sudarytų iš posistemių, projektavimą. Sprendžiant verslo kompiuterizavimo problemas, neretai prireikia platesnės perspektyvos, kuri apžvelgtų sistemą kaip kelių mažesnių sistemų visumą. Jau pats didelių sistemų prigimtinis sudėtingumas reikalauja tokio kūrimo proceso, kuris skaidytų jas į posistemius, atitinkančius tam tikrą kiekį projektų, kurių kiekvienas yra atskira taikomoji programa ar modulis, susiję tarpusavio ryšiais.

Darbą sudaro dvi tarpusavyje glaudžiai susijusios dalys – teorinė ir praktinė. Teorinėje dalyje analizuojami sistemų išskaidymo į posistemius metodai taikant universalų procesą: funkcinis sistemos išskaidymas ir objektinės orientacijos sistemų inžinerija.

Praktinę projekto dalį sudaro sistemos Aplinkos apsaugos inspekcijos teritorinių padalinių posistemių (AAIP) projektas. Įvertinus situaciją galima teigti, kad praktinė projekto dalis labai aktuali aplinkosaugos srities kompiuterizacijai Lietuvoje, kadangi kol kas nėra sukurtos interneto prieigos prie duomenų bazių, saugančių aplinkosaugos duomenis. Tokia prieiga būtų patogi įvairių tipų vartotojams, dirbantiems su aplinkosaugos duomenimis, o dalis duomenų galėtų būti prieinama ir plačiajai visuomenei informavimo tikslais.

Problema. Projekto idėja nėra nauja, tačiau Aplinkos apsaugos inspekcijos teritoriniai padaliniai neturėjo priemonių, leidžiančių betarpiškai naudotis bendra duomenų baze. Dėl šios priežasties projektas buvo labai aktualus norint pasiekti didesnį darbo našumą bei efektyvumą, išvengti gana nemažų laiko sąnaudų. Be to, toks problemos sprendimas pasirodė pigiausias variantas ne tik kūrimo, bet ir produkto palaikymo atžvilgiu.

Darbo tikslas. Praktinis šio magistrinio darbo tikslas buvo sukurti interneto prieigą nutolusiems miškų sistemos objektams (urėdijoms, girininkijoms, aplinkos apsaugos inspekcijoms ir kt.). Tokiu būdu siekta pagreitinti daugelio institucijų bei organizacijų, susijusių su miškų ūkiu, darbą. Projektas buvo susijęs su didesniu projektu – Lietuvos miškų išteklių integruota informacine sistema. Darbe kurtas projektas apėmė Aplinkos apsaugos inspekcijos teritorinių padalinių veiklos funkcijas.

Siekiant kuo geriau įgyvendinti šį darbą, buvo atlikta išsami programinės įrangos projektavimo metodologijų analizė, kurios metu išnagrinėti jau egzistuojančių metodų taikymo panašumai bei skirtumai, nustatyti privalumai. Projektas realizuotas remiantis geresne ir perspektyvesne metodika.

Darbo metodologija. Mokslinės literatūros studijavimas, programinės įrangos projektavimo bei kūrimo įrankių įvaldymas, sistemų projektavimo metodikų taikant universalų procesą (funkcinio sistemos išskaidymo ir objektinės orientacijos sistemų inžinerijos) analizė.

Analizuota literatūra neapsiriboja vien spausdintiniais leidiniais. Projekto metu buvo daug naudotasi ir interaktyviais informacijos šaltiniais, užtikrinančiais ne tik lengvesnį jų prieinamumą, bet ir naujesnius duomenis.

AAI informacinės sistemos projekto realizavimo priemonių pasirinkimui buvo svarstyta keletas alternatyvų: PHP, .NET ir Java technologijos [10], [11]. Dėl plataus kūrimo, palaikymo priemonių ir komponentų pasirinkimo, lengvai pritaikomos atvirojo kodo koncepcijos, nepriklausomumo nuo platformos buvo pasirinkta Java technologija. Teorinėms žinioms gilinti bei įsisavinti dar buvo naudojamasi literatūra internetinėms prieigoms kurti, kurioje nagrinėjama kliento ir serverio architektūra, servletų, *Java Server Pages*, *Enterprise Java Beans* technologijos, aptariamos darbo su duomenų bazėmis naudojant Java ypatybės [9].

Sistemos architektūrai projektuoti buvo studijuojamos UML projektavimo priemonės. Susipažinta su pagrindiniais interneto sistemų modeliavimo stereotipais [6], detaliai nagrinėti internetinių prieigų architektūros komponavimo metodai ir navigacijos struktūros apibrėžimo priemonės [12]. Pati sistemos architektūra buvo kuriama naudojantis IBM kompanijos produktu *Rational Rose*.

Sistemos architektūrai projektuoti buvo analizuojami tradiciniai ir objektinės orientacijos sistemų inžinerijos metodai. Medžiaga surinkta iš tarptautinių mokslinių konferencijų bei simpoziumų [3], [7]. Taip pat remtasi mėnesinio elektroninio žurnalo „Rational Edge“ straipsniais [4], [5], [8]. Darbe pateikti pavyzdžiai iš praktinės magistrinio projekto realizacijos dokumentacijos [1].

Padarius modeliavimo metodų analizę, buvo suformuluotas posistemių projektavimo metodas, kuris pradiniam išskaidymui naudoja funkcinius, o toliau – objektinių metodų principus. Gauta posistemių architektūra užtikrina lanksčias posistemių priklausomybes ir daugkartinio naudojimo galimybes.

Darbo uždaviniai:

1. Išnagrinėti didelių sistemų projektavimo ypatumus, jų išskaidymo į posistemius metodikas, išanalizuoti metodikų privalumus bei trūkumus.
2. Sukurti interneto prieigą Aplinkos apsaugos inspekcijos teritoriniams padaliniams pritaikant įsisavintas teorines žinias.

Darbo struktūra ir trumpa charakteristika.

Darbą sudaro įvadas, trys dėstymo dalys, išvados, literatūros sąrašas, terminų ir santrumpų žodynas. Priede pridedamas darbo tema X-oje tarpuniversitetinėje magistrantų ir doktorantų konferencijoje publikuotas straipsnis.

Pirmoje dalyje aprašoma didelių sistemų sukūrimo poreikių analizė, programų projektavimo technologijų apibrėžimas, sistemos architektūros modeliavimo priemonės bei pagrindžiamas jų pasirinkimas.

Antra dalis skirta detalesniam posistemių projektavimo nagrinėjimui. Teorinės žinios pritaikytos praktiniams pavyzdžiams, kurie pateikiami UML diagramomis, sudarančiomis eksperimentinės Lietuvos miškų instituto informacinės sistemos Aplinkos apsaugos inspekcijos teritorinių padalinių posistemio projektą.

Trečioje dalyje pristatoma projekto eksperimentinė realizacija, pateikiamas naudojimosi sukurta programine įranga aprašymas bei sistemos kokybės analizė.

Išvadose pateikiami esminiai darbo rezultatai.

Darbas naudingas programinės įrangos architektūros projektuotojams, studentams bei visiems kitiems, besidomintiems didelių sistemų projektavimo ir posistemių išskyrimo metodais naudojant UML kalbą ir UML CASE įrankius. Praktinė darbo dalis naudinga besidomintiems interneto prieigų kūrimu; Aplinkos apsaugos inspekcijos darbuotojams; Lietuvos urėdijų, girininkijų ir Lietuvos aplinkos apsaugos ministerijos darbuotojams bei kitiems suinteresuotiems asmenims (LVMI ar LŽŪU studentams ir kt.).

Sistemos vartotojai. Pagrindiniai projekto vartotojai – Lietuvos aplinkos apsaugos inspekcijos padalinių darbuotojai, taip pat urėdijų, girininkijų, Lietuvos aplinkos apsaugos ministerijos darbuotojai ir kiti suinteresuoti asmenys, prižiūrintys regioninius miškus, stebintys jų būklę, atsakingi už leidimų kirsti tam tikrus miškų sklypus išdavimą bei šių leidimų laikymosi priežiūrą.

Informacija apie projekto užsakovą. Projekto užsakovas – Lietuvos Valstybinio Miškų Instituto projektavimo grupė. LVMI (įsteigtas 1950 m.) yra pagrindinis miškotyros centras Lietuvoje. Jis sprendžia miškininkystės mokslo ir miškų ūkio gamybos Lietuvoje problemas, turi didelę eksperimentinę bazę. Institutas teikia plačią metodinę paramą miškų ūkio gamybai, veiksmingai padeda taikyti praktikoje mokslo naujoves, nuolat konsultuoja gamybos specialistus ir miškų savininkus jiems rūpimais klausimais. Ši mokslinė organizacija taip pat yra mokslinės ir specialiosios miškininkystės literatūros rengimo ir leidimo centras Lietuvoje.

Publikuoti straipsniai. Šio darbo tema buvo parengtas straipsnis [2], publikuotas X-os tarpuniversitetinės magistrantų ir doktorantų konferencijos pranešimų medžiagos leidinyje. Straipsnio kopiją galima rasti šio darbo 1 priede.

1. PROJEKTAVIMO METODŲ IR TECHNOLOGIJŲ ANALIZĖ

1.1. Didelių sistemų sukūrimo poreikių analizė

Didelių sistemų projektavimui būdinga „visumos“ perspektyva, kas projektuojant sistemą koncentruojamasi tik į architektūros požiūriu būtinus elementus [3]. Tačiau kiekvienas posistemis privalo atspindėti detales, nes jis įgyvendina pagrindinės sistemos aspektus. Galima sakyti, kad iš posistemių sudarytos sistemos kūrimo procesas remiasi „iš viršaus žemyn“ (pagrindinė sistema suteikia turinį kiekvienam posistemiui) ir „iš apačios į viršų“ principais (kiekvienas posistemis realizuoja tam tikrą sistemos aspektą).

Vadinasi, pagrindinė sistema taip pat yra priklausoma nuo kiekvieno ją išpildančio posistemio. Visumos negalima apibrėžti be jos dalių techninių smulkmenų, o dalių negalima apibrėžti nesuprantant visumos. Tai parodo sistemos bei posistemių tarpusavio sąryšį. Todėl jų kūrimas turėtų būti vykdomas lygiagrečiai. Vis tik dažniau vyrauja tendencija spręsti sudėtingas problemas „iš viršaus žemyn“, kadangi didelę problemą lengviau spręsti suskaidžius ją į aibę mažesnių ir tuo pačiu lengviau išsprendžiamų. Kiekvienas posistemis turi savo aktorius ir panaudojimo atvejus, apibrėžiančius posistemio reikalavimus ir kartu atitinkančius bendros sistemos reikalavimus.

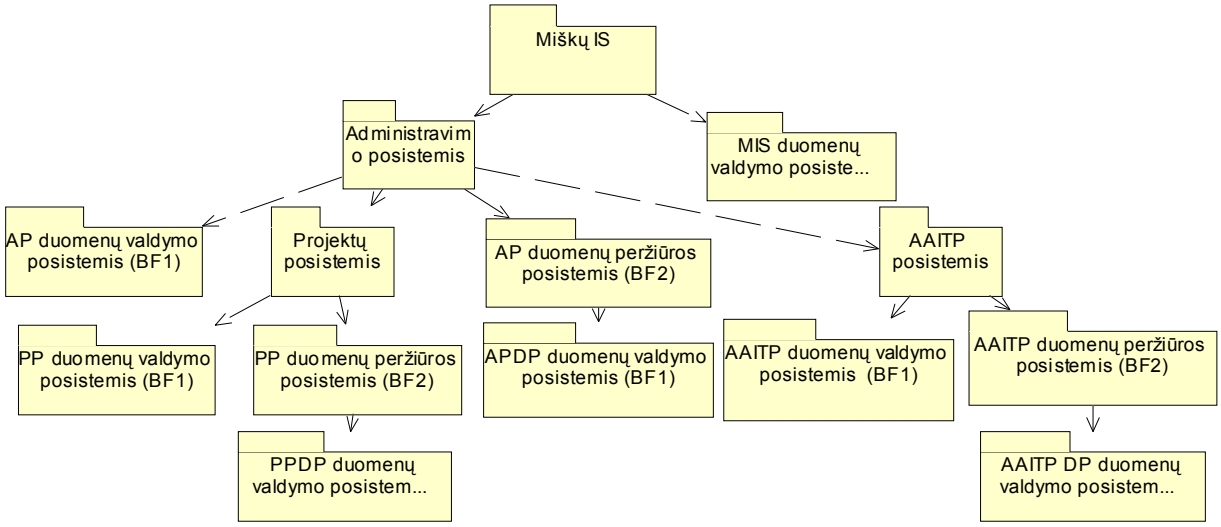
Didelių sistemų kūrime yra rizika, jog galima pamiršti atsižvelgti į tai, kaip suskaidytų problemų sprendimai įtakoja bendrą sprendimą. Susitelkus į mažų dalių kūrimą ir nuolat nesiejant jų su visuma, kūrimo pabaigoje gali tapti neįmanoma sujungti jų į bendrą sistemą. Kita problema – turint daug „mažų dalių“ ir norint jas sujungti (pavyzdžiui, integruojant senas sistemas), reikia apriboti sistemą atsižvelgiant tai, ką galima padaryti iš tų mažų dalių.

1.2. Projektavimo metodų apžvalga

Projektuojant sistemų sistemą ir sumodeliavus projekto detales, kuriami modeliai. Svarbu suvokti modelių paskirtį ir skirtumus tarp jų. Veiklos modelis yra aukščiausio lygio, apibrėžia bendrą sistemos kontekstą. Žemesnio lygmens modelis apbrėžia pačią sistemą – jos bendrą informaciją ir funkcionalumą. Galiausiai išskiriami sistemos posistemiai. Atsižvelgiant į sistemos dydį ir sudėtingumą, posistemiai gali būti išskaidyti į smulkesnes dalis, kurios savo ruožtu, taip pat dar gali būti skaidomos. Tokio skaidymo tikslas – valdomas sistemos apibrėžimas, kuris leistų aiškiai suprojektuoti bendrą sistemą.

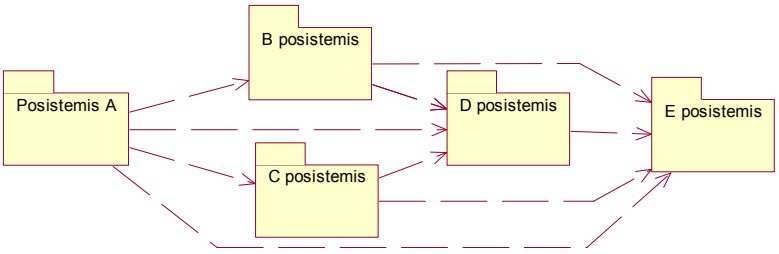
Struktūrinėje metodologijoje skaidant sistemą identifikuojamos bendros funkcijos (1.1 pav., BF1 ir BF2). Analizuojant posistemių išskyrimo pavyzdžius, buvo pastebėta, kad posistemius galima išskirti, analizuojant sistemos elgseną – pavyzdžiui, klasių modelyje rasti pasikartojančias klasių priklausomybių

sekas arba operacijų naudojimo scenarijus sekų diagramose. Funkcinio išskaidymo metu identifikuotos funkcijos glaudžiai susiejamos su viršesnio lygmens komponentais, o tai labai padidina komponentų priežiūros apimtį.



1.1 pav. Nagrinėjamos sistemos architektūra taikant funkcinį požiūrį

Objektiniame modeliavime posistemių išskyrimas remiasi tokiais kriterijais: maksimalus vidinis posistemio glaudumas ir minimalus jo susietumas su išore. Tačiau literatūroje nėra nurodyta, kaip praktiškai išskirti posistemius pagal šiuos kriterijus. Dažniausiai projektuotojai remiasi intuicija. Paprasčiausias sistemos išskaidymas yra jos padalijimas pagal funkcines veiklos sritis (1.2 pav.), tačiau toks išskaidymas gali būti neefektyvus, jei posistemiuose yra pasikartojančios elgsenos grupių.



1.2 pav. Posistemių vaizdas po architektūros projektavimo

Realizacijos etape posistemiai virsta programiniais komponentais. Reikia pastebėti, kad posistemių samprata objektiniame projektavime skiriasi nuo struktūriniuose metoduose naudojamos sampratos, kurioje sistemos posistemiai ir komponentai susideda iš žemesnio lygmens posistemių ar komponentų. Objektinėje metodologijoje vietoje agregavimo ryšio yra priklausomybė arba naudojimo ryšys (kartais jis žymimas stereotipu <<*use*>>): aukštesnio lygmens komponentai naudoja žemesnio lygmens komponentus; konkretus komponentas realizuojamas ir instaliuojamas vieną kartą, o naudojamas daugelį kartų, todėl supaprastėja jo priežiūros, keitimo, tobulinimo procesai.

Lyginamoji metodų analizė. Tradicinis sistemų inžinerijos metodas modeliuoti sudėtingas sistemas – atlikti funkcinį išskaidymą tam, kad būtų galima nustatyti pagrindines sistemos funkcijas ir, tai įvykdžius, pritaikyti tą pačią metodiką didesnių funkcijų suskaidymui į mažesnes. Šis metodas nukreipia sistemos apibrėžimą labiau inžinerinės, o ne vartotojo perspektyvos link ir gali sąlygoti technologiškai sudėtingas, nebūtinai vartotojo poreikius atitinkančias sistemas.

Antras didelių sistemų projektavimo metodas – objektinių sistemų inžinerija. Pastaroji koncentruojasi ties iš posistemių sudarytos sistemos specifikavimo ir skaidymo problemomis. Objektinė metodologija [6] apibrėžia tai, ko vartotojas tikisi iš sistemos (kaip ji veiks daugelyje skirtingų situacijų), ir sukuria panaudojimo atvejų modelį, apimantį šiuos scenarijus, bei klasių modelį, valdantį informaciją ir funkcionalumą.

Šie du UML modeliai suteikia pakankamai informacijos sukurti sistemą, suskaidytą į posistemius, kurių kiekvienas turi savo panaudojimo atvejus ir klases (pagrindiniai abiejų metodų privalumai ir trūkumai pateikti 1.1 lentelėje). Tačiau nėra tiksliai apibrėžtos metodikos, kaip tuos posistemius išskirti. Universaliame projektavimo procese pateikti keli apibendrinti posistemių struktūros variantai (funkcinė; trijų lygių architektūra, išskiriant sąsajos, veiklos paslaugų ir duomenų paslaugų posistemius; daugiasluoksnė), tačiau detalesnių rekomendacijų, kaip išskirti pakartotino naudojimo posistemius, nėra. Būtent dėl šios priežasties projekto metu buvo nuspręsta projektavimo metu sudaryti kitokį metodą, panaudojantį abiejų metodikų stipriąsias puses ir užtikrinantį lengvesnį sistemos plečiamumą bei palaikymą.

1.1 lentelė. Projektavimo metodų palyginimas

Metodas	Privalumai	Trūkumai
Funkcinis išskaidymas	Aiškumas, taikymo lengvumas	Posistemių priklausomybė nuo žemesnio lygmens posistemių, sunkus modifikavimas
Objektinis modeliavimas	Projektavimo principai leidžia pasiekti pakartotiną panaudojimą. Lengvesnė sukurtos programinės įrangos priežiūra, palaikymas ir plečiamumas	Nėra tiksliai apibrėžtos posistemių išskyrimo metodikos, sunkesnis jų išskyrimas

1.3. UML projektavimo priemonės

Net ir labai paprastos programinės įrangos realizavimas be projektavimo priemonių gali tapti gana sudėtinga užduotimi. O didelių sistemų įgyvendinimas be jų sunkiai įsivaizduojamas. Modeliavimas padeda lengviau suprasti sistemos detales, susidaryti tikslų jos vaizdą. Modeliavimo pasirinkimas turi didžiulę įtaką sprendžiamos problemos supratimui ir paties sprendimo formai. Internetinės taikomosios programos, kaip ir kitos programinės įrangos sistemos, yra pavaizduojamos pasitelkiant modelius [12]: panaudojimo atvejų (*use case*), realizacijos (*implementation model*), išdėstymo (*deployment model*) ir kt. Išskirtinis internetinių programų modelis yra tinklalapio žemėlapis, kurį sudaro abstraktus puslapių bei navigacijos kelių, galimų esamoje sistemoje, rinkinys.

Sprendžiant kaip ir ką modeliuoti, labai svarbu nustatyti abstrakcijos ir detalizavimo lygmenį, rasti optimalų variantą sukuriant rezultatą, kuriuo bus patenkinti modelio naudotojai. Paprastai geriausia ir naudingiausia modeliuoti sistemos artefaktus, t. y. tas realaus gyvenimo esybes, kurios bus sukurtos ir kuriomis bus manipuliuojama siekiant realizuoti galutinį produktą – išbaigtą sistemą. Programų tarnybinės stoties vidaus komponentų modeliavimas ar interneto naršyklės detalių modeliavimas neduos laukiamų rezultatų ir nepalengvins sistemos architektų bei projektuotojų darbo. Tinklalapių modeliavimas, nuorodų tarp jų išskyrimas, dinaminio turinio, pateikiamo juos sudarant ir perduodamo klientui analizė yra labai svarbi. Tinklalapiai, nuorodos tarp jų, dinaminio turinio elementai serverio bei kliento pusėje – tai ir yra tie sistemos artefaktai, kuriuos privalu modeliuoti ir analizuoti.

Norint atlikti modeliavimą, reikia sistemos elementus susieti su modelių elementais. Lengviausiai tai padaroma su nuorodomis, kurios automatiškai virsta ryšiais tarp modelio elementų. Tinklalapiai (interneto puslapiai) gali būti suprantami kaip klasės loginiame modelio pjūvyje. Tinklalapius apibrėžus kaip klases, visos kodo funkcijos, naudojamos juose automatiškai virsta klasių operacijomis. Tinklalapio lygmenyje naudojami programavimo kodo kintamieji yra paverčiami klasės atributais. Problema iškyla kuomet tinklalapyje yra naudojama kodo operacijų aibė, kuri veikia tarnybinės stoties pusėje, formuojant dinaminį tinklalapio turinį, bei visiškai skirtingi kliento pusės kodo elementai (pvz., *JavaScript*), valdantys sąsajos elementus ir atliekantys pirminę įeinančių duomenų kontrolę arba dalinį tinklo navigacijos sistemos valdymą. Tai sukelia sumaištį, kadangi vienu atveju turime dar nesuformuotą internetinį puslapį tarnybinės stoties pusėje, kitu atveju – jau kitą aktyvią dalį, kuri yra pateikiama kliento dalyje. Taigi paprastas interneto puslapio susiejimas su UML klase nepadės projektuotojams geriau suprasti kuriamos sistemos.

UML kūrėjai numatė atvejį, kada jų pateikta išbaigta UML semantika nesugebės padengti specifinės nagrinėjamų sistemų objektų bei komponentų aibės. Tam tikslui yra naudojamas formalus UML išplėtimo

mechanizmas kuris yra naudojamas standartinės UML semantikos išplėtimui. Šis mechanizmas leidžia nustatyti stereotipus, žymėtąsias reikšmes bei apribojimus, kurie gali būti taikomi modelio elementams.

Stereotipas yra elementas suteikiantis galimybę nustatyti naują semantinę reikšmę modelio elementui. Žymėtosios reikšmės yra pagrindinės reikšmių poros kurios gali būti susietos su modelio elementu taip priskiriant reikšmę šiam elementui. Apribojimus sudaro taisyklės, kurios garantuoja kuriamo modelio teisingumą. Šios taisyklės gali būti išreikštos kaip laisvos formos tekstas arba formalizuotos panaudojant objektų apribojimų kalbą (*OCL – Object Constraint Language*).

Bendroji internetinių taikomųjų programų architektūra yra sudaroma analizuojant naršyklių, tinklo bei programų tarnybinės stoties galimybes. Naršyklės „pareikalauja“ tinklalapių iš tarnybinės stoties. Kiekvieną tokį tinklalapį galima apibūdinti kaip turinio bei jo suformavimo taisyklių rinkinį, išreikštą pasinaudojant HTML. Kai kurie tinklalapiai naudoja kliento pusės programinius kodus, kurie yra interpretuojami naršyklės. Šie kodo elementai suteikia papildomą dinamiką tinklalapio elgsenai bei jo pavaizdavimui, kurį vykdo naršylė. Dažnai tokie metodai yra naudojami dinamiškai keisti turinio vaizdavimą, papildomas valdykles (apletus ir kt.) naudojamus internetiniame puslapyje. *Web* aplikacijos naudotojas stebi pateikiamą tinklalapio informaciją ir sąveikauja su sistema pasinaudodamas jam suteikiamais valdymo elementais – informacijos laukais arba nuorodomis į kitus sistemos tinklalapius. Bet kurie vartotojo atlikti veiksmai įtakoja sistemos darbą ir keičia jos veiklos būseną.

Žiūrint į *Web* puslapį iš vartotojo pusės galima teigti, jog tai visuomet bus HTML pagrindu suformuotas dokumentas. Serverio pusėje tas pats puslapis gali keisti savo turinį ir būseną pagal įvairius scenarijus, kurie yra sukuriami pasinaudojant scenarijų rašymo metodologijomis bei kalbomis.

Modeliuojant *Web* tinklalapius kiekvienas interneto puslapis yra susiejamas su UML komponentais. Komponentu yra laikoma fizinė ir keičiama kuriamos sistemos dalis. Modelio realizacijos pjūvis (komponentų pjūvis) atvaizduoja sistemos komponentus ir ryšius tarp jų. Internetinių taikomųjų programų atveju, šiame pjūvyje yra atvaizduoti visi sistemą sudarantys tinklalapiai ir ryšiai tarp jų (šie ryšiai sudaromi panaudojant nuorodas). Viename lygmenyje *Web* sistemos komponentų diagrama atrodo kaip tinklalapio žemėlapis.

Tinklalapio komponentai atvaizduoja tik fizines sąsajos dalis, jie nėra tinkami vaizduoti bendradarbiavimą bei sąveiką tinklalapio viduje. Toks abstrakcijos lygmuo yra labai svarbus sistemos kūrėjams ir programuotojams ir turi būti modelio dalimi. Kiekvienas *Web* puslapis yra klasė atvaizduojama UML dizaino pjūvyje (loginiame pjūvyje). Atvaizduojant tinklalapyje esančius metodus kaip klasės operacijas, o jų naudojamus kintamuosius kaip klasės atributus yra susiduriama su serverio bei kliento pusės metodų nesuderinamumu. Vienas iš šios problemos sprendimo būdų yra panaudoti

stereotipus taip kiekvienam klasės atributui arba operacijai nurodant atitinkamą priklausomybę. Tačiau tuomet modeliavimas iš paprasto ir lengvai suprantamo tampa komplikuotu ir sunkiai suvaldomu.

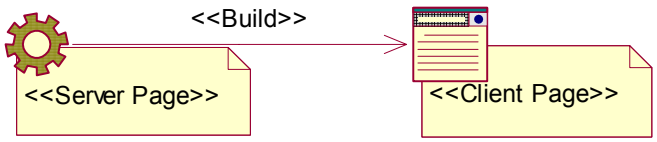
Daug geresnis šios problemos sprendimas yra „interesų išskaidymas“. Mąstant logiškai, *Web* puslapio elgesys serverio ir kliento pusėje yra visiškai skirtingas. Kai tinklalapis yra apdorojamas serverio pusėje, jam suteikiama prieiga prie serverio resursų (failų sistemos, duomenų bazių ir kt.). Tas pats puslapis sudarytas iš srauto HTML komandų kliento pusėje yra visiškai kitoks, jei nagrinėsime jo galimus veiksmus bei resursų panaudojimą. Kliento pusėje tinklalapis gali naudoti naršyklės resursus ir elementus, pasinaudojant dokumento objektų modeliu (*DOM – Document Object Model*) ir kiekvienu jame specifikuotu įskiepu, Java apletu arba ActiveX komponento valdykle. Taipogi galimi ryšiai su kitais tuo metu aktyviais puslapiais, esančiais kitoje tinklalapio karkaso dalyje arba naršyklės esybėje.

Išskiriant tokius interesus, galima modeliuoti serverio pusės aspektus kaip vieną klasę, o kliento pusės – kaip kitą. Toks išskyrimas į dvi dalis atliekamas pasinaudojant UML stereotipais: serverio tinklalapiu ir kliento tinklalapiu.

Serverio tinklalapis vaizduoja *Web* puslapį, kuriame yra programinio kodo vykdomo serverio pusėje. Šis programinis kodas sąveikauja sus serverio resursais (duomenų bazėmis, failais ir kt.). Šio stereotipo objektai yra programiniam kode aprašytos funkcijos, o atributai – tinklalapio lygmens globalūs kintamieji.

Kliento tinklapio esybė yra HTML pagrindu suformuotas *Web* tinklalapis. Kaip ir bet kuris kitas tinklalapis, jis yra duomenų, įvykių apdorojimo mechanizmo ir vaizdavimo elementų rinkinys. Šio stereotipo metodai yra kliento dalies funkcijos, išreikštos tinklalapyje, o atributai – tinklalapio lygmens kintamieji. Kliento tinklalapis gali turėti asociacijų tiek su kliento, tiek ir serverio tinklapiais.

Taip sukuriama nauja UML semantika modeliuojamiems elementams. Klasė su nurodytu stereotipu gali būti išskirta panaudojant ikonas.

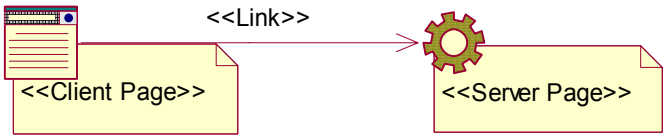


1.3 pav. *Build* ryšys tarp UML interneto stereotipų

Interneto puslapių atveju, stereotipas paženklina klasę kaip loginės tinklalapio elgsenos abstrakciją serverio arba kliento pusėje. Abi šios klasės yra susiejamos kryptiniu ryšiu, nukreiptu iš serverio pusės tinklalapio į kliento pusės tinklalapį ir suteikiant ryšiui stereotipą *<<build>>*, kadangi serverio dalies tinklalapis suformuoja kliento dalies tinklalapį (1.3 pav.). Kiekvienas dinaminis tinklalapis (toks tinklalapis, kurio turinys yra suformuojamas serverio pusėje vykdomų aplikacijų) yra konstruojamas kartu

su serverio puslapiu. Dažniausiai vienas serverio puslapis suformuoja vieną kliento pusės internetinį tinklalapį, tačiau yra sistemų kuriuose vienas serverio tinklalapis formuoja net kelis kliento dalies puslapius.

Paprasčiausias ryšys tarp tinklalapių yra nuoroda. Nuoroda *Web* aplikacijoje atlieka navigacijos kelio sistemoje vaidmenį. Toks ryšys tarp tinklalapių modeliuose yra vaizduojamas pasinaudojant stereotipu <<*link*>>. Tokio stereotipo asociacija visuomet prasideda kliento dalies tinklalapyje ir rodo į serverio arba kliento dalies internetinį puslapį (1.4 pav.).



1.4 pav. *Link* ryšys tarp UML interneto stereotipų

Nuorodos yra įdiegiamos į sistemą kaip tinklalapio paklausą, o *Web* puslapiai yra modeliuojami komponentai, kuomet visi jie vaizduojami realizacijos pjūvyje (*Implementation View*). Nuorodos pagrindu sudaromas ryšys tarp serverio ir kliento klasių (su nurodytu stereotipu <<*build*>>) ir paprasta nuoroda sudarytas ryšys (su stereotipu <<*link*>>) yra ekvivalentai, kadangi tiek vienu, tiek ir antru atveju šie ryšiai yra tinklalapio pareikalavimas.

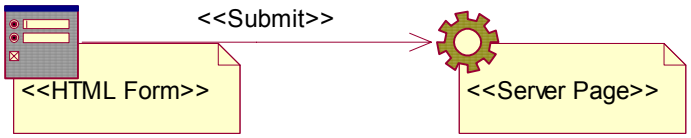
Žymių reikšmės (*tagged values*) yra naudojamos aprašyti parametrams, kurie yra perduodami per nuorodas užklausiant internetinio tinklalapio. Prie nuorodos asociacijos pridėta žymių reikmė yra sąrašas parametrų vardų, kurie yra naudojami formuojant užklaustą internetinį tinklalapį.

Stereotipų panaudojimas labai palengvina internetinių tinklalapių programinių kodų bei jų naudojamų ryšių modeliavimą. Serverio puslapių stereotipo klasės operacijos tampa atitinkamo tinklalapio metodais, o atributai – to tinklalapio lygmens globaliais kintamaisiais. Kliento puslapių operacijos tampa kliento pusėje galimais metodais, o atributai – atitinkamo lygmens kintamaisiais. Taip serverio pusės tinklalapiams yra modeliuojami santykiai su serverio pusės resursais, o kliento pusės – su naršyklės elementais ir Java apletais.

Vienas iš didžiausių privalumų, pastebimų naudojant klasių stereotipus modeliuojant loginę tinklalapio elgseną yra tas, kad jo sąveika ir bendradarbiavimas su serverio pusės komponentais yra išreiškiama taip pat kaip bet kuri kita tik serverio dalies komponentų sąveika. Serverio puslapio stereotipu pažymėta klasė yra tiesiog kita klasė dalyvaujanti sistemos veiklos logikos modeliavime. Daugiau konceptualiame lygmenyje šios serverio lygmens klasės dažniausiai nustatinėja sistemos būsenos valdiklius, taip aktyvuodamos įvairius sistemos veiklos objektus norint sukurti tam tikrą, nuo sistemos einamos būsenos priklausantį, veiklos rezultatą, kurio naršyklės pagalba pareikalavo sistemos naudotojas.

Kliento pusėje vykstančių sąveikų ir bendradarbiavimo modeliavimas gali būti šiek tiek komplikuotas. Tai įtakoja plati aibė technologijų, kurios gali būti panaudotos šioje dalyje. Paprasčiausias kliento pusės tinklalapis yra HTML dokumentas, kuris sudarytas iš valdančiojo navigacinio kodo bei turinio informacijos. Naršyklė naudotojui atvaizduoja tinklalapį remdamasi HTML formatavimo instrukcijomis, pateikiamomis jame su kartais panaudojamais vaizdavimo stiliais. Vaizdavimo stilių stereotipas gali būti išskirtas ir suformuota atskira klasė, su priklausomybės ryšiu nuo kliento pusės tinklalapiu. Tačiau šio stereotipo klasės yra išskiriamos gana retai.

Pagrindinis duomenų įvedimo mechanizmas internetiniuose puslapiuose yra forma. Formos yra apibrėžiamos HTML tinklalapiuose panaudojant <*form*> žymę. Kiekvienai formai yra nurodomas tinklalapis, į kurį yra siunčiami surinkti duomenys. Formą sudaro aibė informacijos įvesties elementų, visi jie yra išreiškiami HTML žymėmis. Dažniausiai naudojamos žymės yra <*input*>, <*select*> ir <*textarea*>. HTML žymė <*input*> gali būti daugiareikšmė įvesties elementų formos atžvilgiu, nes ją galima vaizduoti kaip teksto įvedimo lauką, daugybinės arba vienetinės pasirinkties laukas, paslėptas laukas ir kt. Modeliuojant formas sukuriamas dar vienas klasių stereotipas <<*form*>>. Formos stereotipo klasė neturi operacijų, kadangi bet kokia operacija aprašyta formos žymių viduje yra tinklalapio, o ne formos lygmens. Formos duomenų įvedimo elementai yra formos stereotipo klasės atributai. Visiems jiems, atsižvelgiant į duomenų įvesties tipą, yra priskiriamas atitinkamas stereotipas. Formos stereotipo klasė gali turėti ryšių su apletais, jei šie yra naudojami kaip informacijos įvesties elementai. Kiekviena forma taipogi turi ryšį su serverio dalies tinklalapiu, kuriame yra apdorojami formos surinkti duomenys ir kuris yra nurodytas kaip formos duomenų siuntimo tikslas. Šis ryšys yra parašomas kaip <<*submit*>> (1.5 pav.).



1.5 pav. *Submit* ryšys tarp UML interneto stereotipų

Viena ar kelios JavaScript bibliotekos gali būti aprašytos kaip <<*JavaScript*>> stereotipo klasės. Jų metodai tampa operacijomis, o globalūs kintamieji – atributais.

Interneto aplikacijose vis dažniau yra imami naudoti karkasai (*frames*), kurie leidžia vienu metu naudotojui matyti kelis tinklalapius ir šie tinklalapiai gali bendradarbiauti ir sąveikauti tarpusavyje keisdamiesi duomenimis bei aktyvuodami įvairius objektus. Naujausios naršyklių versijos leidžia sąveikauti net skirtinguose naršyklės vienetuose esantiems tinklalapiams. Tai yra vykdoma naudojant dinaminio HTML programinį kodą.

Ar naudoti karkasą internetinėje aplikacijoje turi nuspręsti sistemos architektai bei programuotojai. Jei karkasai yra naudojami, jie turi būti išreikšti ADM (*Application Domain Modeling*). Modeliuojant karkasų naudojimą atsiranda dar dvi stereotipinės klasės – <<*frameset*>> ir <<*target*>> – bei asociacijos stereotipas <<*target link*>>. Karkasų aibės klasė yra tiesiogiai susieta su HTML <*frameset*> žyme. Joje yra nurodomi kliento dalies tinklalapiai. Tikslo klasė ir yra tinklalapis, kuris yra talpinamas karkaso viduje. Jei šie tinklalapiai bendradarbiauja tarpusavyje, tai turi būti sukurtas tikslinės nuorodos ryšys. Kaip karkaso dalis yra vaizduojama ne tik vienos naršyklė vaizduojami tinklalapiai, bet ir kelių naršyklių tinklalapiai, kurie tarpusavyje yra susiję tam tikrais sąveikos mechanizmais. Prie ryšių tarp karkaso ir jo elementų gali būti nurodomos žymimos reikšmės, kurios nusako eilutę bei stulpelį, į kuriuos yra nukreipiamas vaizduojamasis elementas.

Pasinaudojus visais šiais klasių stereotipais bei asociacijomis tarp jų galima labai detaliai ir išsamiai atlikti internetinių taikomųjų programų modeliavimą, neatsižvelgiant į jų struktūros sudėtingumą ir lankstumą. Toks UML profilis interneto svetainėms projektuoti gali būti naudojamas bet kokio tipo internetinėms aplikacijoms.

1.4. Technologijų analizė

Renkantis kūrimo platformą, svarstyta keletas alternatyvų: PHP, .NET ir Java technologijos. Išstudijavus literatūrą [10], [11] buvo apibendrinti technologijų privalumai ir trūkumai, pateikti toliau (1.2 lentelė). Galutinis pasirinkimas buvo Java technologijos dėl plataus kūrimo, palaikymo priemonių ir komponentų pasirinkimo, lengvai pritaikomos atvirojo kodo koncepcijos, nepriklausomumo nuo platformos. PHP nusileido dėl silpnesnio objektinio programavimo koncepcijos, didesnio primityvumo. .NET technologija – dėl kūrimo, palaikymo priemonių monopolio priklausančio „Microsoft“ ir dėl mažesnio nemokamų komponentų pasirinkimo.

Skyriuje pateikiama tik pasirinktų naudoti projekto eksperimentinėje realizacijoje Java technologijų analizė. Toliau apžvelgiama:

- servletai,
- *Java Server Pages* (JSP) dinaminiai puslapiai,
- *JavaBeans* komponentai,
- *Struts* karkasas,
- *Struts* karkaso architektūra,
- programų konstravimo aplinkos *Eclipse* pritaikymas *Struts* karkasui.

1.2 lentelė. Programavimo technologijų charakteristikos

	PHP	**.NET**	**Java**
Palaikomos kalbos	PHP	Palaiko apie 25 skirtingas programavimo kalbas (*JScript*, *VBScript* ir t. t.)	Java
Programavimo principai	Objektinis arba procedūrinis programavimas, taip pat dažnas abiejų principų mišinys	Priklausomai nuo programavimo kalbos, taikomas ir objektinis, ir procedūrinis programavimo principai	Objektinis programavimas
Priklausomybė nuo platformos	Pritaikyta beveik kiekvienai platformai	Geriausiai pritaikyta „Microsoft“ operacinėms sistemoms	Nepriklausoma
Funkcionalumas	Pasiekiamas per atskirus modulius (grafikos įrankius, susijungimo su DB tvarkykles ir t. t.)	Gaunamas sudėtingo kodo pagalba	Jį sąlygoja pakartotinio panaudojimo komponentai ir žymos
Atviras kodas	✓	Dauguma paketų priklauso „Microsoft“ korporacijai	✓
Patikimumas	Turi nežymių trūkumų, tačiau dėl atviro kodo aptiktos klaidos greitai ir efektyviai pataisomos	Kadangi tai ne atviro kodo produktas, programų paketo klaidų negali pataisyti nepriklausomi programuotojai	Patikima
Sparta	Interpretuojama programavimo kalba, sparta nusileidžia daugumai kompiliuojamųjų	Dauguma paketo programavimo kalbų kompiliuojamos	Tik iš dalies kompiliuojama (programos kodas iš pradžių kompiliuojamas į baitinį kodą, o paskui virtuali mašina tą kodą interpretuoja)
Palaikomumas		Galimybė naudoti daug programavimo kalbų programų palaikymą daro sudėtingu	Jį sąlygoja pakartotinio panaudojimo komponentai

Servletai. Java technologija, pritaikoma internetui, yra servletai – programų serverį praplečiantys komponentai. Prieš atsirandant JSP technologijai, serverio komponentai atlikdavo skaičiavimus, o vartotojo informaciją generuodavo spausdindami rezultatus ir jų atvaizdavimą formuojančius HTML elementus. Nepaisant to, kad skaičiavimai gali būti visiškai susimaišę su duomenų išvedimu, programos kodas turi būti perkompiliuojamas ir įdedamas (*deploy*) apdorojimui. Mažų programų atveju tai neatrodo didelė problema, tačiau kai informacinę sistemą sudaro didelė apimtis panašaus kodo, pakeitimų atlikimas sukelia daug darbo sąnaudų.

***Java Server Pages* dinaminiai puslapiai.** Su JSP, atsirado galimybė skaičiavimus įterpti tarp HTML elementų, taip sumažinant vaizdavimo ir skaičiavimų tarpusavio ryšius. Taip pat nebereikia kiekvieną kartą kompiliuoti ir įkelti kodo, atlikus pakeitimus.

Java veiklos logika, naudojant skriptletus puikiai įsikomponuoja HTML dokumente, aprašančiame duomenų atvaizdavimą, tačiau išlieka nepatrauklus programos kodo įsiterpimas tarp žymomis (*tags*) aprašomų vaizdavimo elementų [12]. Dėl to didelio projekto atveju parašytas kodas sunkiau skaitomas ir pasigendama didesnio veiklos logikos ir vaizdavimo logikos atskyrimo. Norint išspręsti skaičiavimų ir atvaizdavimo problemą, reikia JSP naudoti kartu su architektūriniais šablonais.

***JavaBeans* komponentai.** Galima dalį kodo iš JSP perkelti į Java komponentus. Vieni iš tokių komponentų yra *JavaBeans* klasės [9]. Tai standartiniai objektai kurie susideda iš:

- klasės kintamųjų,
- metodų *setXxx*( ) ir *getXxx*( ) kiekvieno kintamojo reikšmei gauti ir nustatyti,
- konstruktoriaus be parametrų.

Naudojant šias klases, sumažėja veiklos logikos perduodant duomenis tarp JSP puslapių, kurių vienas yra HTML forma, o kitas tos formos duomenis apdorojantis komponentas. *JavaBeans* komponentai gali būti naudojami ir su *Struts* karkasu.

Sudėtingesnės architektūros *Enterprise JavaBeans* veiklos objektai atlieka operacijas su duomenų objektais ir pateikia duomenis atvaizdavimui – redaguoti ir išvesti vartotojui. EJB pagalba pasiekiami duomenys, užuot programos tekste rašius SQL užklausas. Taip sukuriamas turtingas programos veiklos sluoksnis. Tačiau panaudojant šį architektūrinį modelį, vis tiek išlieka skaičiavimų ir duomenų atskyrimo problema.

***Struts* karkasas.** Tai vienas iš pirmųjų karkasų pritaikytų JSP, ir:

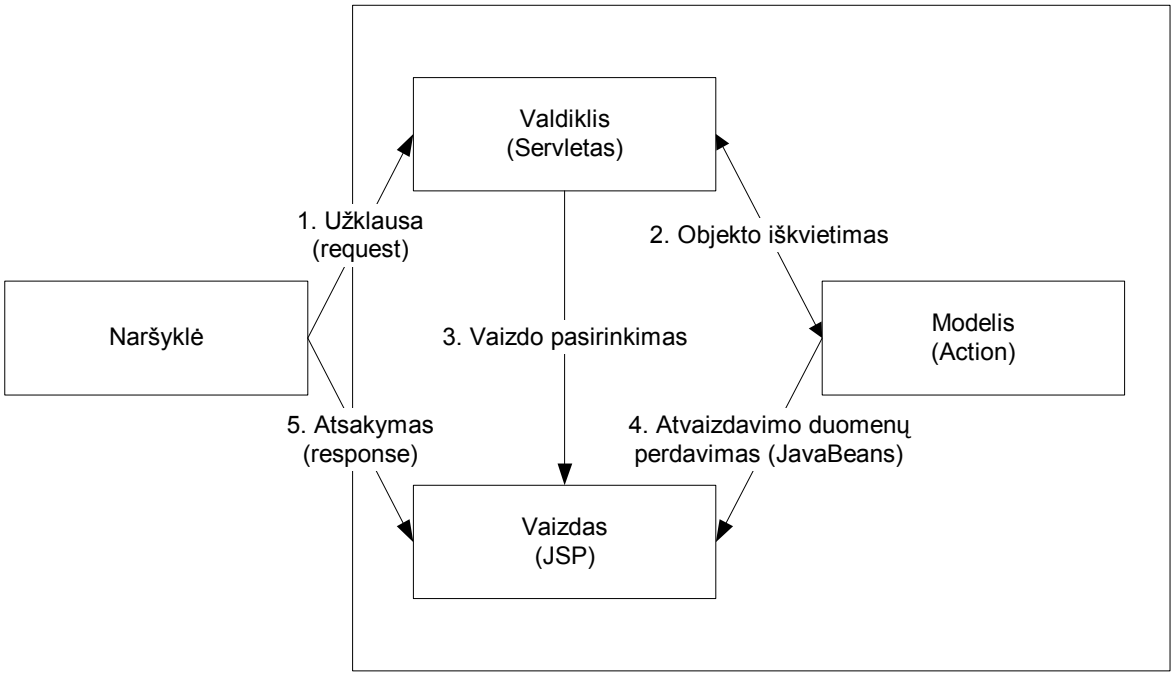
- pateikia žymų bibliotekas, atliekančias logines, iteracijų operacijas, bei darbą su duomenimis, kas išsprendžia JSP įskaitomumą. *Bean* ir *logic* žymų bibliotekos naudojamos šioms problemoms išspręsti;

- naudoja MVC architektūrą, kuri atskiria skaičiavimo ir pateikimo logiką [11].

JSP puslapyje, nenaudojant *Struts*, tiesiogiai kuriamas šio tipo kintamasis ir kreipiamasi į jo metodus, kad išvesti duomenis, o naudojant *Struts* žymų bibliotekas, pakanka kaip žymos parametrą perduoti kintamojo vardą, taip visiškai išvengiant Java kodo JSP puslapyje.

Žinoma, ir šiame architektūriniame modelyje galima naudoti skriptletus įterpiant veiklos logiką į JSP puslapius, tačiau tokia praktika nėra rekomenduojama.

***Struts* karkaso architektūra.** Programų sistemos kūrimui pasirinktas *Struts* karkasas, todėl darbui su juo buvo atlikta detali šio paketo analizė. Žemiau pavaizduota komponentų sąveikos diagrama (1.6 pav.) paaiškina tarpusavio sąveikas [9].



1.6 pav. *Struts* komponentų sąveika

Kliento programa (naršyklė), kreipiantis į valdiklio funkcijas atliekantį servletą, šis iškviečia reikiamą veiklos komponentą (pvz., *JavaBean*) ir parenka atvaizdavimo komponentą JSP, į kurią perduodami atvaizdavimui skirti duomenys iš *JavaBean*. Suformuoti rezultatai atvaizduojami vartotojui.

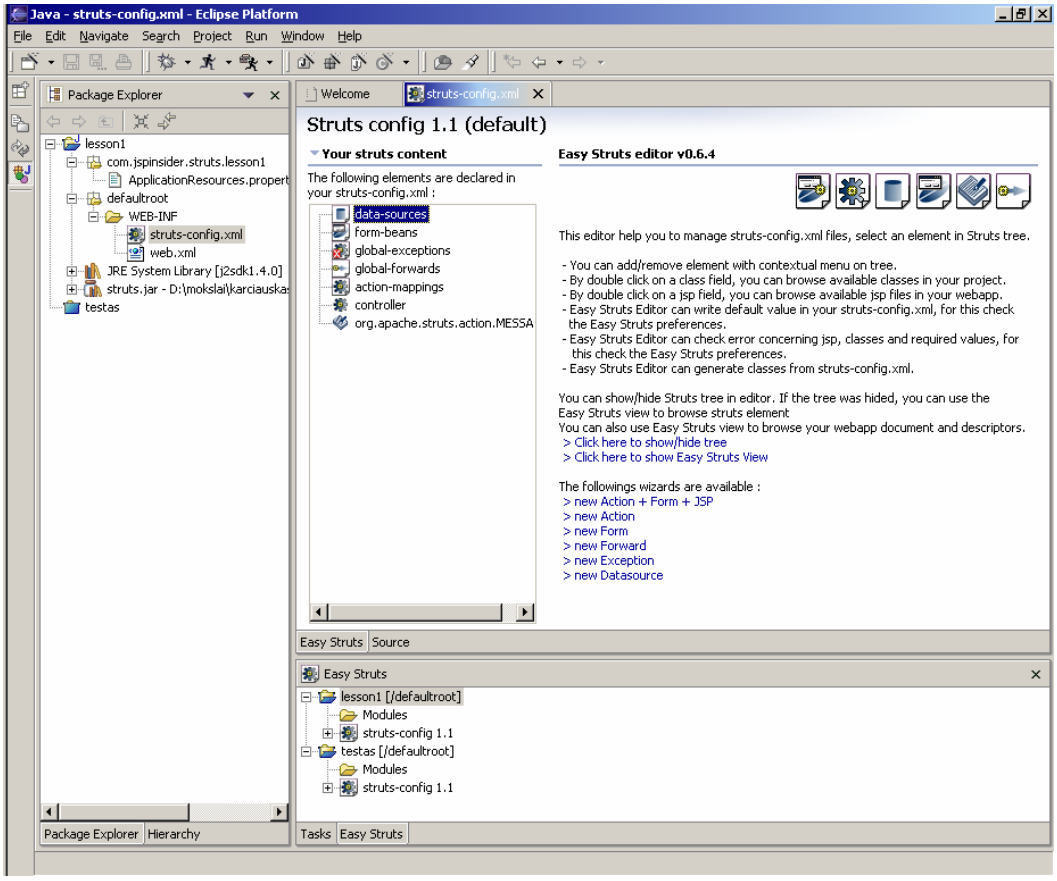
Patalpinus programų sistemą servletų konteineryje pirmiausia nuskaitomas šis failas ir *Struts* servletų apdorojimui priskiriamos tam tikro tipo užklausos. Dažniausiai užklausos, kurios baigiasi *.do* priskiriamos *Struts* apdorojimui.

Programų konstravimo aplinkos *Eclipse* pritaikymas *Struts* karkasui. *Eclipse* programavimo aplinka (IDE) pasirinkta dėl universalumo ir plataus pritaikymo, bei paplitimo tarp vartotojų. Tai yra

atvirojo kodo produktas pasiekiamas adresu <www.eclipse.org>. *Struts* komponentų generavimo galimybė kuo puikiausiai įmanoma pasiekti naudojant šią programavimo aplinką.

Sukūrus naują Java projektą, reikia jo nustatymuose (*project→properties→easy struts→add struts support*) pažymėti apie *Struts* projekto kūrimą ir pasirinkti karkaso nustatymus kuriamam produktui.

Atlikus pakeitimus, į projekto bibliotekas įkeliamos *Struts* bibliotekos, projekto katalogų struktūra pritaikoma kuriamam projektui – *Struts* informacinei sistemai. Sukuriamas struts-config.xml failas. Projekto medyje išsirinkus šį failą pateikiamas automatizuoto kūrimo meniu.



1.7 pav. *Eclipse Easy Struts* praplėtimo automatizavimo pasirinkimas

1.7 paveiksle matyti, kad apačioje pateikiamas praplėtimo peržiūros medis. Kairėje atvaizduojami struts-config.xml failo elementai: *data-sources, form-beans, action-mappings* ir kt. Automatizavimo požiūriu svarbiausia yra pateikti žyniai. Naudojantis jais, nesunkiai galima gauti sugeneruotą konfigūracinį failą struts-config.xml bei komponentų karkasus.

Programavimo aplinka puikiai praneša apie vykusias klaidas, joje patogu naršyti po projekto resursus. Projekto valdymas apima ir patogią techninės informacijos generavimą (*javadoc*). Programos paleidžiamieji failai išsaugomi projekto konfigūracijoje.

Struts karkaso panaudojimas ypač patogus – *Easy Struts* praplėtimas pateikia komponentų generavimo žynių rinkinį, priemones specifinės architektūros klaidų sekimui, patogią navigaciją.

1.5. Analizės išvados

Sprendžiant verslo kompiuterizavimo problemas, neretai prireikia platesnės perspektyvos, kuri apžvelgtų sistemą kaip kelių mažesnių sistemų visumą. Jau pats didelių sistemų prigimtinis sudėtingumas reikalauja tokio kūrimo proceso, kuris skaidytų jas į posistemius, atitinkančius tam tikrą kiekį projektų, kurių kiekvienas yra atskira taikomoji programa ar modulis, susiję tarpusavio ryšiais.

Darbe išnagrinėti sistemų skaidymo į posistemius metodai: funkcinis sistemos išskaidymas ir objektinės orientacijos sistemų inžinerija. Remiantis nustatytais jų privalumais bei trūkumais, nutarta pradiniam skaidymui naudoti paprastą ir aiškų funkcinį išskaidymą, o tolesnėse projektavimo iteracijose pritaikyti objektinių metodų principus ir identifikuoti daugkartinio naudojimo komponentus.

Kadangi kūrimas vyks pagal objektinės inžinerijos principus, sistemos architektūros projektavimui pasirinkta unifikuota modeliavimo kalba UML ir *Rational Rose* CASE įrankis. Pateikta pagrindinių UML elementų apžvalga bei įsitikinta, jog galima labai detaliai ir išsamiai atlikti net ir sudėtingų internetinių taikomųjų programų modeliavimą.

Renkantis kūrimo platformą projekto realizavimui, svarstyta keletas alternatyvų: PHP, .NET ir Java technologijos. Išstudijavus literatūrą ir atlikus lyginamąją analizę, pasirinktos Java technologijos dėl stipraus objektinio programavimo, nepriklausomumo nuo platformos, atvirojo kodo prieinamumo ir gausaus įrankių, bei komponentų pasirinkimo.

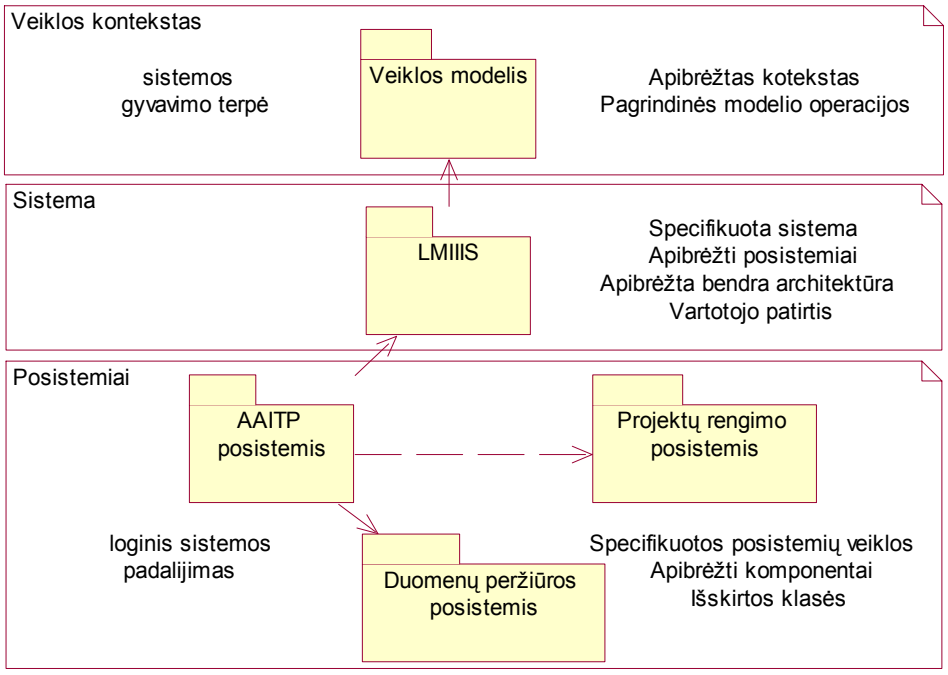
2. POSISTEMIŲ PROJEKTAVIMO METODAS UNIVERSALAUS PROJEKTAVIMO PROCESE

Skyriuje pristatomas siūlomas projektavimo metodas, naudojantis abiejų pirmame skyriuje aprašytų didelių sistemų projektavimo metodikų stipriąsias puses.

Dauguma toliau darbe naudojamų pavyzdžių yra paimti iš praktinės darbo dalies [1], todėl toliau universalaus projektavimo proceso metodai pristatomi Aplinkos apsaugos inspekcijos teritorinių padalinių programinės įrangos kontekste.

2.1. Posistemių projektavimas

Projektuojant sistemų sistemos architektūrą prisireikia sukurti ne vieną modelį. Todėl labai svarbu suprasti tų modelių hierarchinę struktūrą bei jų tarpusavio ryšius. Hierarchinės struktūros viršuje yra veiklos (verslo) modelis (2.1 pav.), apibrėžiantis įmonės veiklos sritį, o tuo pačiu ir kuriamos sistemų sistemos kontekstą. Detalesnis yra pačios sistemos modelis, apibrėžiantis bendrą jos funkcionalumą ir padedantis projektuotojui išskirti atskirų posistemių funkcijas. Ir galiausiai, turint veiklos ir sistemos modelius, galima pradėti skaidyti sistemą į posistemius, išskiriant jų komponentus, klases ir kitus komponentus. Priklausomai nuo sistemos sudėtingumo, posistemiai gali būti skaidomi į mažesnius sektorius, o šie – dar į mažesnes dalis ir t. t.



2.1 pav. Modelių hierarchinė struktūra

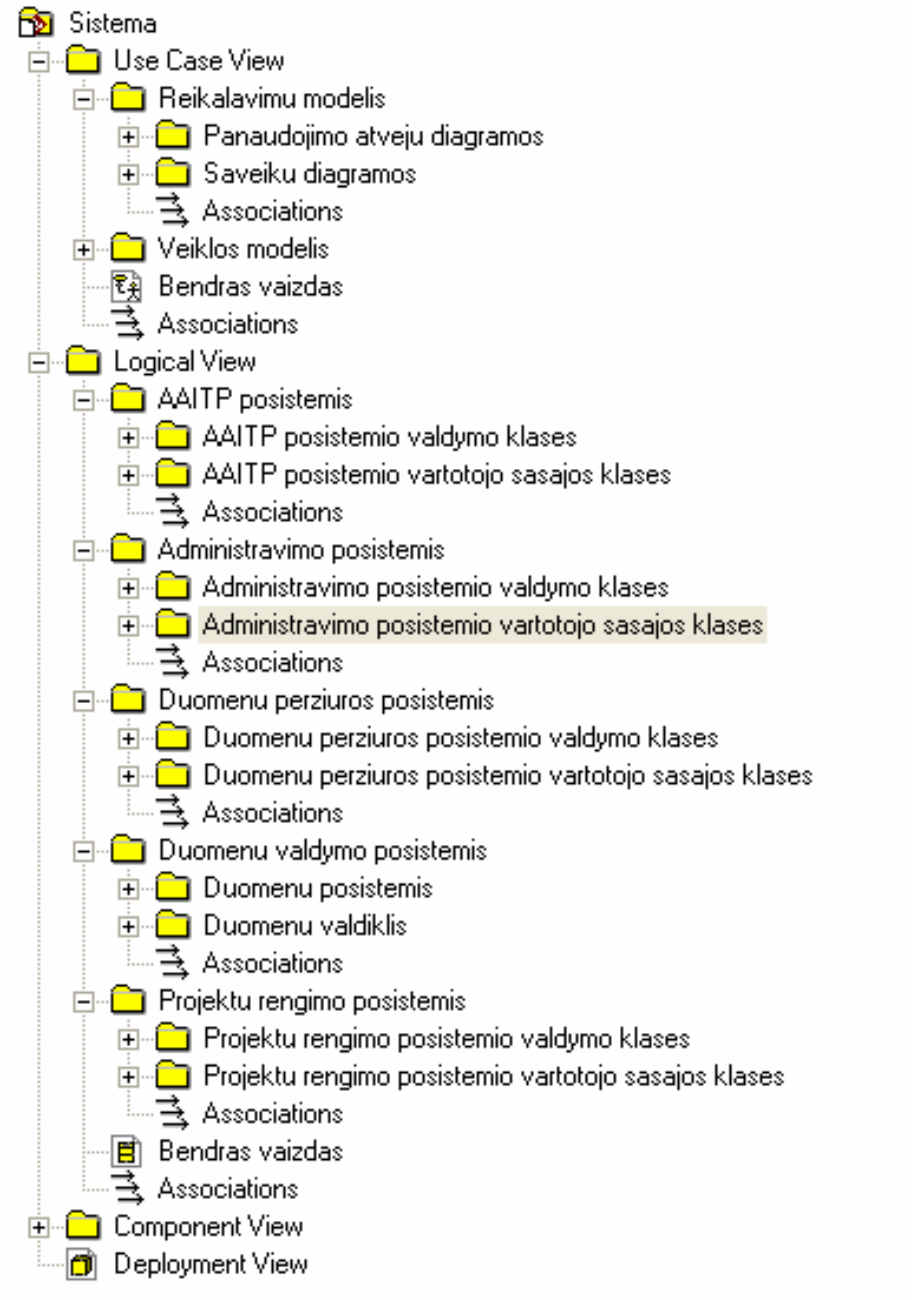
Posistemių išskyrimo iš sistemos modelio ir jų projektavimo procesas gali būti suskaidytas į tokius pagrindinius žingsnius [3]:

1) sistemos konteksto modelio sudarymas;
2) sistemos aukšto lygio panaudojimo atvejų ir dalykinės srities klasių diagramų braižymas:
    a) sistemos panaudojimo atvejų apžvalga,
    b) dalykinės srities klasių išskyrimas,
    c) sistemos panaudojimo atvejų detalizavimas veiklos diagramų pagalba,
    d) veiklos diagramų, praplėstų veiklos objektų reikalavimais, sudarymas;
3) veiklos diagramų performavimas į sekų diagramas,
4) dalykinės srities klasių diagramų pergrupavimas siekiant suformuoti posistemio klasių bendrą vaizdą, pagrįstą klasių tarpusavio sąryšiais,
5) sekų diagramos sudedamųjų dalių apibendrinimas siekiant nustatyti posistemio panaudojimo atvejus;
6) posistemio diagramos sudarymas panaudojant panaudojimo atvejų aktorius.

Pirmieji du algoritmo žingsniai detaliai detalizuoja sistemos modelį. Kiti trys – jį praplečia, su tikslu išskirti posistemius ir apibrėžti kiekvieno posistemio panaudojimo atvejus ir dalykinės srities objektus.

Šis algoritmas taip pat gali būti taikomas ir sistemos modelio sudarymui iš veiklos modelio. Tokiu atveju svarbu sudaryti veiklos modelį taip, jog vėliau būtų galima iš jo išskirti panaudojimo atvejus ir pradines dalykinės srities klases sistemos modeliui.

Suskaidžius sistemą atskirų posistemių architektūrai specifikuoti braižoma visa eilė detalizuotų modelių. Tuo tikslu buvo pasirinkta naudoti UML ir *Rational Rose* CASE įrankį. 2.2 paveiksle pateikta architektūros modelių, sudarytų realizuojant AAIP informacijos sistemą, struktūra.



2.2 pav. Projekto architektūros schema

2.2. Sistemos veiklos konteksto apibrėžimas

Metodika visų pirma rekomenduoja apibrėžti pačią sistemų sistemos gyvavimo terpę, jos kontekstą, kuriame būsimasis produktas bus naudojamas ir palaikomas. Būtent tam ir reikalingas veiklos modelis, kuriame atspindima veiklos anatomija, apibrėžiamos verslo, o tuo pačiu ir būsimos sistemos, ribos ir jo santykis su išoriniais objektais (vykdytojais).

Šios sistemos tikslas –sukurti internetinę prieigą nutolusiems miškų sistemos objektams (urėdijoms, girininkijoms, aplinkos apsaugos inspekcijoms ir kt.).

2.3. Pagrindinės kuriamos sistemos funkcijos

Skyriuje aprašytos pagrindinės kuriamos sistemos funkcijos (2.1 lentelėje) pateikiant jas funkcinių reikalavimų ir jų tikimo kriterijų pavidalu. Taip pat skyriuje pateikiami ir nefunkciniai reikalavimai sistemos panaudojamumui, vykdymo ir veikimo sąlygoms, informacijos saugumui.

2.1 lentelė. Reikalavimų sistemos funkcionalumui sąrašas

Nr.	Reikalavimas	Tikimo kriterijus
1	Sistema turi neįsileisti neregistruotų vartotojų	Sistema turi išvesti pranešimą apie klaidingai įvestus duomenis.
2	Sistema turi užtikrinti visų tipų vartotojų teises atlikti reikiamus veiksmus	Sistemos administratorius registruoja vartotojus ir suteikia jiems slaptažodžius. Sistema besinaudojančių vartotojų teisės skirtingos, todėl suteikiami skirtingi slaptažodžiai.
3	Sistema turi leisti vartotojams atlikti tik tuos veiksmus, kuriuos leidžia jų teisės	Vartotojo teisių apribotos funkcijos pačiam vartotojui gali būti matomos, tačiau neprieinamos (pvz., neaktyvūs saitai ar mygtukai).
4	Sistema gali naudotis tik registruoti vartotojai	Asmuo, siekiantis pasinaudoti sistema, turės įvesti slaptažodį. Informaciją apie galimybę naudotis sistema jis gaus elektroniniu paštu iš administratoriaus, prižiūrinčio sistemą AAIP įdiegimo vietoje.
5	Sistemą turi būti galima modifikuoti sistemos administratoriui	Administratorius turi turėti galimybę tinkamai prižiūrėti AAIP svetainę ir, esant reikalui, padaryti joje pakeitimus.
6	Sistema leidžia registruoti leidimus kirsti mišką ir išveda pranešimą, jei registruojamas leidimas jau egzistuoja duomenų bazėje	Leidimai kirsti mišką registruojami atsižvelgiant į sudarytus kirtimų projektus, užregistruotus AAIP. Leidimas galės būti išduotas tik asmeniui, kurio duomenys neįvesti į AAIP DB. Sistema turi išvesti pranešimą, jei registruojamas leidimas jau egzistuoja duomenų bazėje. Du kartus tas pats leidimas negali būti užregistruotas.
7	Sistema leidžia sudaryti tikrinamų sklypų sąrašus ir išveda pranešimą, jei sudarytas sąrašas jau yra	Tikrinamų sklypų sąrašai sudaromi pasinaudojus paieškos rezultatais, analizuojant miškų būklės ir ankstesnių kirtimų duomenis. Jei duomenų bazėje sudarytas sklypų sąrašas jau egzistuoja, jo užsaugoti negalima. Sistema turi išvesti klaidos pranešimą.

2.1 lentelės tęsinys. Reikalavimų sistemos funkcionalumui sąrašas

Nr.	Reikalavimas	Tikimo kriterijus
8	Sistema leidžia registruoti patikrinimo aktus ir išveda pranešimą, jei patikrinimo aktas jau yra užregistruotas	Atlikus miško sklypo patikrinimą, patikrinimo aktai turi būti registruojami sistemoje. Priklausomai nuo padarytų išvadų, turi būti suteikta galimybė keisti duomenis, susijusius su patikrinimo aktais ir sklypų charakteristikomis. Jei duomenų bazėje patikrinimo aktas jau egzistuoja, jo užregistruoti dar kartą negalima. Sistema turi išvesti klaidos pranešimą.
9	Sistema leidžia atlikti informacijos paiešką ir peržiūrą	Sistema turi užtikrinti tikslią informacijos paiešką ir peržiūrą visiems vartotojams. Priklausomai nuo vartotojų kategorijos, jiems galima peržiūrėti skirtingus duomenis. Galima tikrintų/netikrintų sklypų, girininkų kertamų sklypų, išduotų leidimų kirsti mišką ir tikrinimo aktų paieška bei peržiūra. AAIP inspektoriams turi būti prieinama visa reikalinga informacija.
10	Sistema leidžia parengti ataskaitas	Turi būti užtikrintas tikrinamų sklypų sąrašų, leidimų ir jų pratęsimų, patikrinimo aktų ataskaitų suformavimas.
11	Sistema turi užtikrinti pagalbą	Sistema vartotojui turi pateikti vartotojo vadovą.
12	Sistema turi tikrinti įvedamų duomenų korektiškumą	Sistema turi atlikti bent minimalią įvedamų duomenų kontrolę tuo metu, kai vartotojas duoda vykdymo komandą. Klaidos atveju komanda nevykdoma, o išvedamas pranešimas.

Reikalavimų panaudojamumui sąrašas:

- Sistema turi būti paprasta naudotis
- Sistemoje padarytų klaidų tikimybė turi būti kuo mažesnė
- Vartotojo sąsaja turi būti stabili
- Sistema turi būti lietuviška
- Sistemos pagalba vartotojui turi būti išsami.

Vykdymo savybės:

- užduočių vykdymo greitis – kadangi sistema taikoma nutolusiems vartotojams, užduočių atlikimo greitis labiau priklausys ne nuo pačios sistemos, o nuo vartotojo internetinio ryšio kokybės;
- talpumas – vartotojo kompiuteryje AAIP sistema vietos neužims, kadangi tai internetinė prieiga;

- galimas reikšmių diapazonas – galimos visos matematinės operacijos realių skaičių apibrėžimo srityje;
- pasiekiamumas – nutolę vartotojai sistema galės naudotis bet kuriuo metu.

Reikalavimai veikimo sąlygoms. Šio projekto galutinis produktas – programinė įranga, nesijusi su jokiomis mechaninėmis sistemomis, todėl sistemos veikimo neįtakos gamtinės sąlygos ar jos vartotojų darbo sąlygos. Sistema veiks kiekviename kompiuteryje, turinčiame *Windows* operacinę sistemą, internetinę naršyklę bei internetinį ryšį.

Reikalavimai saugumui:

- konfidencialumas – sistemoje esančiais duomenimis turi turėti galimybę naudotis tik registruoti jos vartotojai. Be to, priklausomai nuo vartotojų poreikių, duomenų pasiekiamumas turi būti skirtingas;
- vientisumas – duomenys turi būti perduodami internetu, todėl jų vientisumą turi užtikrinti TCP/IP protokolas;
- pasiekiamumas – nutolę vartotojai turi turėti galimybę naudotis sistema bet kuriuo metu.

2.4. Pradinių panaudojimo atvejų išskyrimas ir reikalavimų specifikavimas

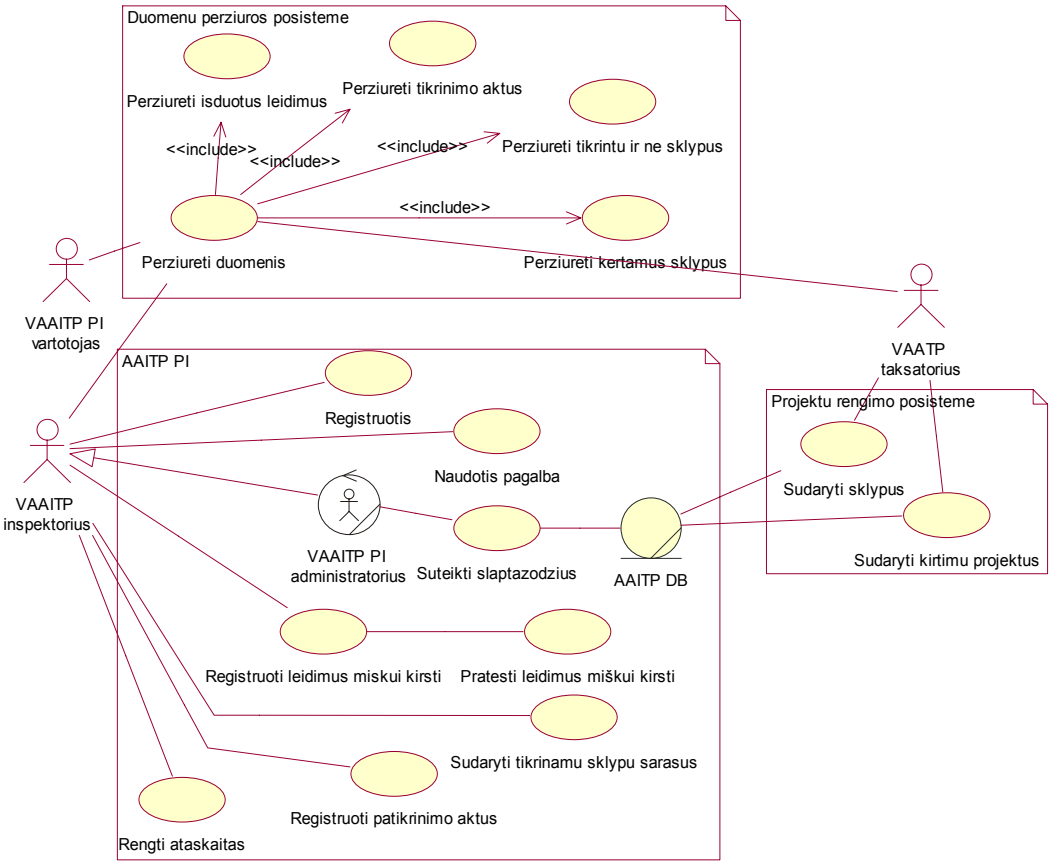
Apibrėžus būsimosios sistemos ribas, būtina išskirti veiklos panaudojimo atvejus (PA) bei sudaryto dalykinės srities klasių modelį. Abi šios diagramos glaudžiai susiję tarpusavyje ir turi iteracinę prigimtį. Iš esmės visada pradedama nuo veiklos panaudojimo atvejų diagramos, susijusios su išoriniais veiklos procesais. Kalbant apie AAI posistemį, jo veiklos panaudojimo atvejai glaudžiai siejasi su LMIIIS (2.3 pav.). Tuomet panaudojimo atvejai detalizuojami (detalizuotos PA diagramos) ir braižoma dalykinės srities klasių diagrama. Galiausiai veiklos diagramos papildomos srities objektais.

Išskiriamus PA pravartu ne tik registruoti, bet ir juos dokumentuoti, t. y. trumpai aprašyti, sudaryti kiekvieno jų specifikacijas. Nors naudojant UML įrankius tai padaryti galima daugeliu būdų, praktika rodo, kad efektyviausiai PA specifikacijos nustatomos ir surenkamos interviu su srities ekspertais (subjektais) metu, sistemos reikalavimų nustatymo etape. Be to, srities ekspertai gali prisidėti ir patys kurdami PA modelius. Pagrindinis šio etapo tikslas – surinkti informaciją apie tai, ko reikia išoriniams vykdytojams iš įmonės arba ką įmonė jiems turi padaryti ir kaip veiklos darbuotojai turi bendradarbiauti, kad pasiektų šį tikslą. Kiekvieno tokių pirminių poreikių rinkinio arba išorinio vykdytojo tikslas diagramoje virsta panaudojimo atveju.

Siekiant išvengti neefektyvių sprendimų ateityje, siūloma atlikti panaudojimo atvejų analizę pasikartojančių scenarijų atžvilgiu ir gauti lanksčią, pokyčių atžvilgiu efektyvią architektūrą, kuri leidžia sumažinti projektavimo, kūrimo bei sistemos priežiūros darbų apimtis.

Dažnai klasės būna pavadintos skirtingais, tačiau gana panašiais vardais, ir jų skaičių galima sumažinti, įvedant apibendrinančias klases, kurias galima sugrupuoti į posistemius. O pakartotinio

panaudojimo požiūriu tikslinga juos išskirstyti į atskirus panaudojimo atvejus, kurie įeina į kitų panaudojimo atvejų sudėtį.



2.3 pav. AAIP panaudojimo atvejų diagrama

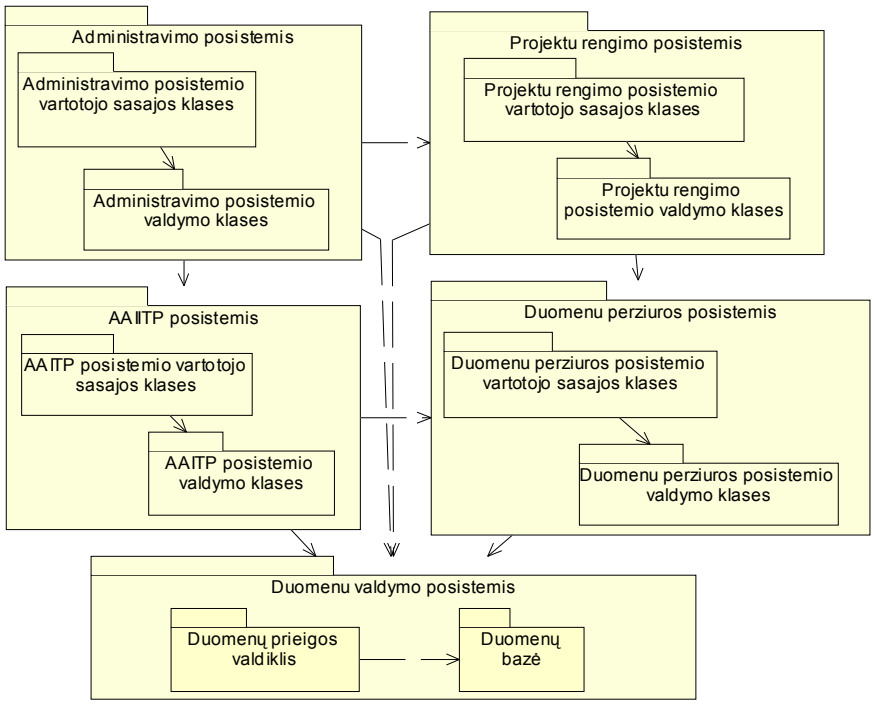
Siūloma išskaidyti sistemą į posistemius iteraciniu būdu, nagrinėjant jos panaudojimo atvejų specifikacijas. Išskaidymas remiasi panašių bei pasikartojančių elementų radimu pradiniuose sistemos panaudojimo atvejų aprašymuose. Tokiu būdu išskirti posistemių panaudojimo atvejai padeda išpildyti bendros sistemos panaudojimo atvejus.

2.5. Loginės architektūros projektavimas

Sukūrus PA sekų diagramas, galima nustatyti tolesnę sistemos loginio išskaidymo struktūrą. Kiti sistemos išskaidymo algoritmo (žr. 2 skyrių) žingsniai sąlygoja iteracinį procesą ir toliau skaido sistemą į mažesnes dalis. Tam reikia peržiūrėti iš naujo veikos panaudojimo atvejų ir dalykinės srities klasių diagramas. Veiklos panaudojimo atvejai toliau detalizuojami, klasių diagrama papildoma metodais, apibrėžtais sekų diagramose, klasės pergrupuojamos siekiant suformuoti naujų posistemių klasių bendrą vaizdą.

Kiekvieną kartą reikia peržiūrėti ir visos sistemos posistemių modelį bei papildyti jį naujai išskirtais elementais. Tie nauji elementai papildo projektuojamo posistemio architektūros specifikacijas, apibrėžia jo funkcionalumą (dalykinės srities klasių diagramos papildymo atveju) ir yra pagrindas tolesniam loginės struktūros skaidymui į mažesnius posistemius. Veiklos PA modelio papildymas vėlesniuose sistemos kūrimo etapuose taip pat dar gali būti naudingas ir kaip testavimo scenarijų rinkinys.

Projektuojant loginę sistemos architektūrą, buvo laikoma, jog posistemiai iš anksto yra nustatyti veiklos modelyje.



2.4 pav. Klasių paketų diagrama

Tuo atveju, jei posistemiai iš anksto nežinomi, visos veiklos diagramos sudaromos „juodos dėžės“ principu; kas reiškia, jog veiklos diagramoje bus tik vienas takelis su projektuojamo posistemio pavadinimu. Tačiau tas takelis vis tiek turi būti detalizuojamas sukuriant objektų srautų ir sekų diagramas, o posistemiai nustatomi ieškant panašumų tarp klasių metodų ir sugrupuojant tas klases kartu į atskirą posistemį. Funkcionalumas, susietas su tomis klasėmis per jų operacijas, apibrėžia naujai išskirto posistemio funkcionalumą. Tokio išskaidymo tikslas yra gauti laisvai susietų posistemių rinkinį su minimaliu (arba be jokio) pertekliniu funkcionalumu. Pritaikius šį procesą AAIP projektavime, gaunamas 2.4 paveiksle pateiktas loginis sugrupuotų klasių paketų vaidas.

3. DEALIZUOTAS APLINKOS APSAUGOS INSPEKCIJOS POSISTEMIŲ PROJEKTAS

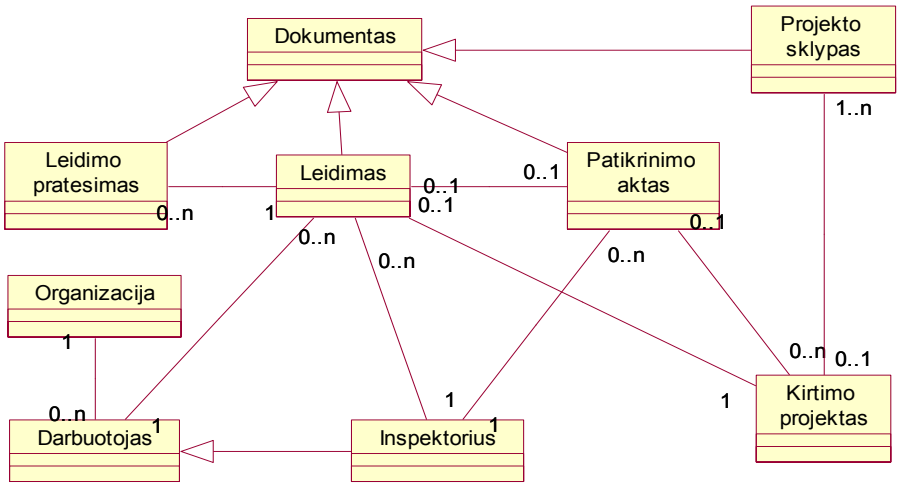
Projekto pagrindas: problema, su kuria susiduria nutolę Lietuvos miškų išteklių integruotos informacinės sistemos (LMIIIS) vartotojai. Iki šiol nėra jokių alternatyvių programinių priemonių, leidžiančių nutolusiems miškų sistemos objektams (urėdijoms, girininkijoms, aplinkos apsaugos inspekcijoms ir kt.) betarpiškai naudotis LMIIIS duomenų baze. Urėdijos keičiasi su miškotvarkos institutu duomenimis, persiųsdamos duomenų bazės kopijas, kurios kol kas apdorojamos pasenusiomis programomis. Aplinkos apsaugos inspekcijos iš viso neturi programinės įrangos ir naudojasi popieriniais dokumentais.

Įdiegus AAIP programinę įrangą, išsispręstų lėto bei neefektyvaus darbo problema, galima būtų sutaupyti darbuotojų laiką bei sumažinti organizacijų išlaidas. Kuriamas projektas apims aplinkos apsaugos inspekcijos teritorinių padalinių veiklos funkcijas, tačiau dalinai juo galės naudotis ir kitų institucijų vartotojai.

Darbe pateikiama tik esminės diagramos ir aprašai, bendrai atspindintys AAI posistemio architektūrą. Išsamų architektūros specifikacijų rinkinį galima rasti projekto galutinės specifikacijos dokumente [1].

3.1. Dalykinės srities klasių modelis

Dalykinės srities klasių modelis yra logiškai sugrupuotų klasių ir jų tarpusavio ryšių rinkinys, pavaizduotas diagramų pavidalu, kur kiekviena klasė žymi aiškią srities informacijos dalį. Šioje proceso stadijoje klasės neturi metodų (gali neturėti ir atributų), bet kiekviena jų turi aiškų rašytinį apibrėžimą (2.4 pav.). Šios klasės atspindi realaus pasaulio informaciją, reikalingą vartotojams, kad šie galėtų atlikti savo užduotis, atspindimas panaudojimo atvejuose. Ta informacija gali būti įvairi, apimti tiek išorinius, tiek ir vidinius sistemos faktorius. Tokios „išorinės“ informacijos pavyzdžiai gali būti įvairūs leidiniai, dokumentai, elektroninės žinutės ar laiškai, rankraščiai. „Vidine“ informacija gali būti laikoma AAIP registruojami patikrinimo aktai, naudojami miško žemėlapiai, sudaromos sutartys ir pan. Tokių klasių identifikavime interviu su srities ekspertais yra kone geriausia (efektyviausia) priemonė.



3.1 pav. AAIP dalykinės srities klasių diagrama

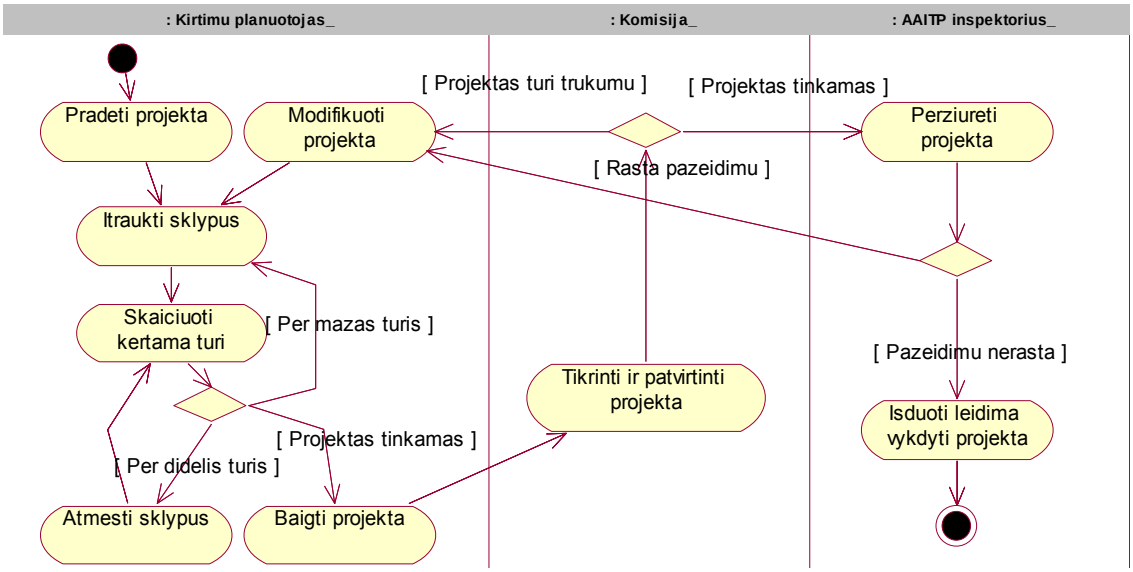
Dalykinės srities klasių modelis naudingas dėl šių priežasčių:

1) padeda identifikuoti ryšius tarp dalykų, kurių tarpusavio ryšių negalima nustatyti iš panaudojimo atvejų modelio;
2) tarnauja kaip pradinės analizės griaučiai ir karkasas vėlesniam klasių kūrimo procesui;
3) yra tikslus sistemos žodyno apibrėžimas iš vartotojo perspektyvos.

3.2. Sistemoje vykdomi procesai

Veiklos diagrama detalizuoja pasirinktą panaudojimo atvejį įvairiais scenarijais. Ji sudaroma iš objektų, veiklų ir įvykių tarpusavio ryšių, kas apima sąveikavimą tarp sistemos ir išorinių vykdytojų. Svarbu pažymėti, jog veiklos diagramos nėra duomenų srautų diagramos – jos parodo įvykių, o ne duomenų srautą. 2.5 paveiksle pateiktas tokios diagramos pavyzdys. Jame kiekvienas takelis atspindi skirtingą išorinį vykdytoją (aktorių) ar veiklos darbuotoją (vidinį aktorių).

Kaip jau minėta anksčiau, dalykinės srities ir veiklos modeliai yra glaudžiai tarpusavyje susiję. Todėl tolesniuose projektavimo etapuose papildyti juos taip pat reikia lygiagrečiai, priklausomai nuo informacijos, esančios antrame iš jų. Pvz., veiklos diagramoje esantys veiklos procesai gali būti papildyti elementais, įtakojančiais tuos procesus (pvz., diagrama gali būti papildyta elementais, nurodančiais kur ir kada reikia papildomų dokumentų tam tikrai veiklai įvykdyti). Tuomet dalykinės srities klasių modelis taip pat turi būti papildytas atitinkamomis klasėmis (jei jame dar nėra tą objektą aprašančios klasės).



3.2 pav. Kirtimų projekto veiklos diagrama

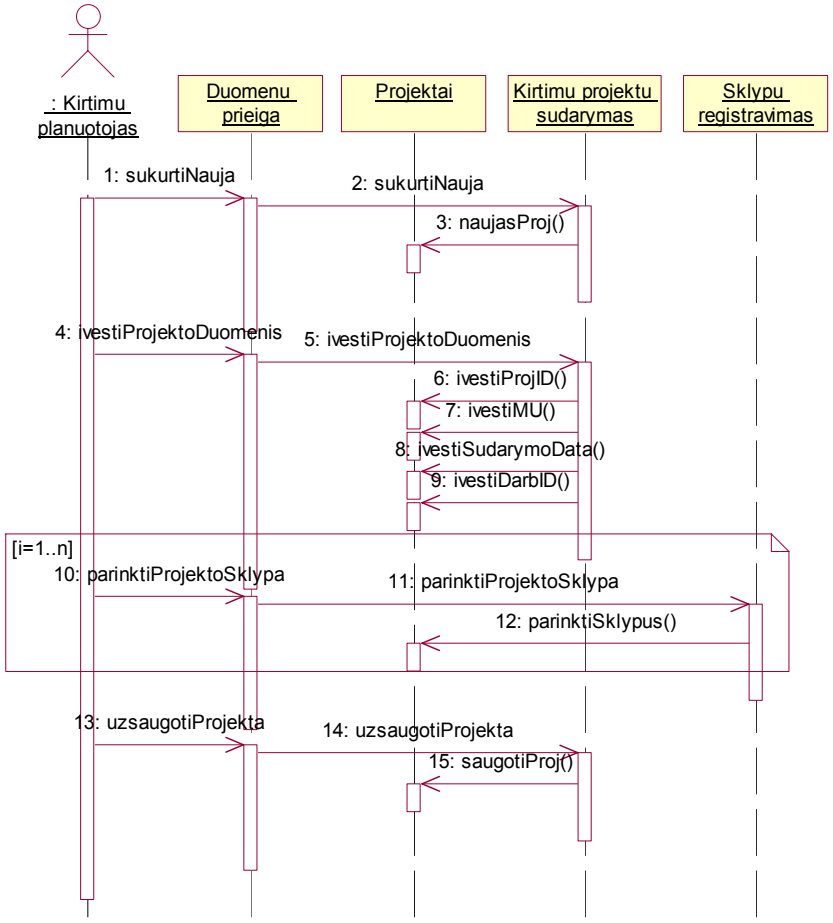
3.3. Sąveikos diagramos

Iki šio skyriaus darbe pateiktos tik nedetalizuotos diagramos, bendrai apibrėžiančios AAI posistemį. Turint aukšto lygio veiklos panaudojimo atvejų, dalykinės srities klasių ir veiklos diagramas, jau galima susidaryti preliminarų posistemio, kaip atskiro vieneto, vaizdą ir pradėti skaidyti jį patį į mažesnes, labiau detalizuotas ir, savo ruožtu, lengviau valdomas dalis. Pirmas žingsnis tokios detalesnio posistemio aprašymo link yra sąveikos (sekų) diagrama.

Apskritai, sistemų projektavimo metu sąveikos diagramos gali būti naudojamos daugelyje modelių: veiklos, reikalavimų, analizės, projekto. Jų pagalba aprašomos sąveikos tarp ne tik tarp klasių objektų, bet ir sistemų, posistemių ar programinių komponentų.

Norint išskaidyti sistemą ar posistemį į mažesnes dalis, reikalavimų modelyje sąveikos diagramos turėtų būti braižomos kiekvienam panaudojimo atvejui. Tokiu būdu PA papildomi scenarijais (pagrindiniais ir alternatyviais), o diagramos parodo aktorių sąveikas su sistema ar posistemiu. Tokių diagramų rinkinį galima naudoti kaip PA specifikaciją. Apibrėžtos operacijos susiejamos su atitinkamomis dalykinės srities klasėmis, aktoriais ar kitomis esybėmis. Kiekvieno metodo veiksmai turi būti aprašyti. Sąveikos su kitais posistemiais jau pradeda apibrėžti aprašomo posistemio interfeiso priklausomybes ir santykį su kitais to paties lygmens sistemų sistemos posistemiais. Sekų diagramoje galima sukurti naujus, dar neklasifikuotus, objektus arba įtraukti jau egzistuojančių klasių egzempliorius.

2.6 paveiksle pateikta diagrama susieta su 2.5 paveikslo veiklos diagramos pirmuoju takeliu ir atspindi pagrindinių AAI posistemio dalių tarpusavio sąveikas bei santykį su veiklos darbuotoju.

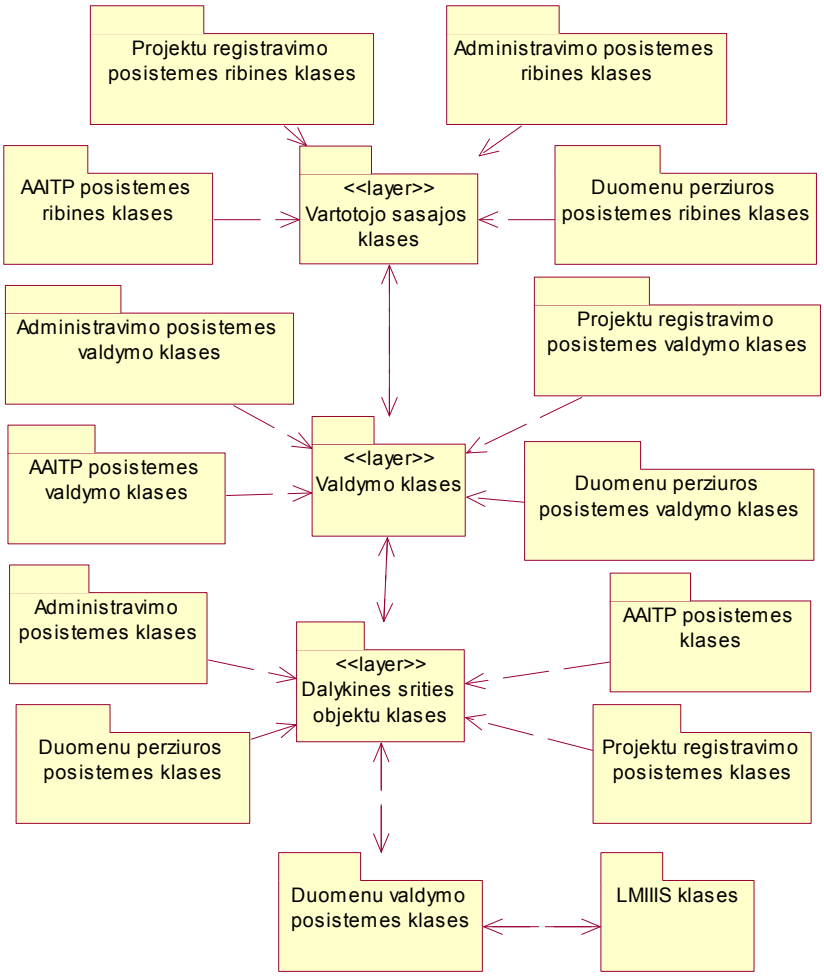


3.3 pav. Projektų sudarymo sekų diagrama

3.4. Sistemos klasių modelis

Projektuojant loginę sistemos architektūrą, buvo laikoma, jog posistemiai iš anksto yra nustatyti veiklos modelyje (3.4 pav.).

Antroji klasių diagrama (3.5 pav.) yra skirta duomenų bazės realizavimui, čia pagrindinis dėmesys yra skiriamas duomenų objektams ir ši diagrama gali būti transformuojama į DB, kur kiekvieną lentelę atitinka diagramos klasė, o įrašų laukus – klasės atributai. Įrašų duomenų tipai bus tikslinami sekančiuose etapuose (realizacijos lygmenyje), tuomet gali būti nustatomi įrašų tipai ir koreguojama pati duomenų klasės struktūra. Kiekviena iš klasių atitinka duomenų bazės lentelę.



3.4 pav. Klasių paketų diagrama

Toliau pateikiama dalykinės srities klasių diagramos kiekvienos klasės paskirtis:

Leidimai. Lentelė, skirta leidimų kirsti mišką registracijai. Joje saugomi antraštės duomenys.

Leid_eilute. Leidimas išduodamas projektui, kurį gali sudaryti daug kirtimui paruoštų sklypų. Todėl šioje lentelėje saugomi kiekvieno projekto sklypo duomenys (sklypo ID, miško iškirtimo ir medienos išvežimo terminai).

Leid_pratesimai. Saugoma informaciją apie pratęstą leidimą kirsti mišką (nauji miško iškirtimo ir medienos išvežimo terminai).

Patikrinimo_aktai. Lentelė, skirta privačių valdų individualių miškotvarkos projektų patikrinimo aktų informacijos saugojimui. Saugomi antraštės duomenys.

Patikrinimo_akt_eilute. Kiekvieną patikrinimo aktą gali sudaryti daug projekte esančių ir patikrintų sklypų. Lentelėje saugomi duomenys apie kiekvieno sklypo taksacinę charakteristiką, aptiktus pažeidimus ir ūkinės priemonės panaikinimo faktą (loginis laukas).

Projektai. Lentelėje saugoma taksatorių registruojama informacija apie sudarytus kirtimo projektus (sudariusio taksatoriaus ID, sudarymo data, suminiai laukai).

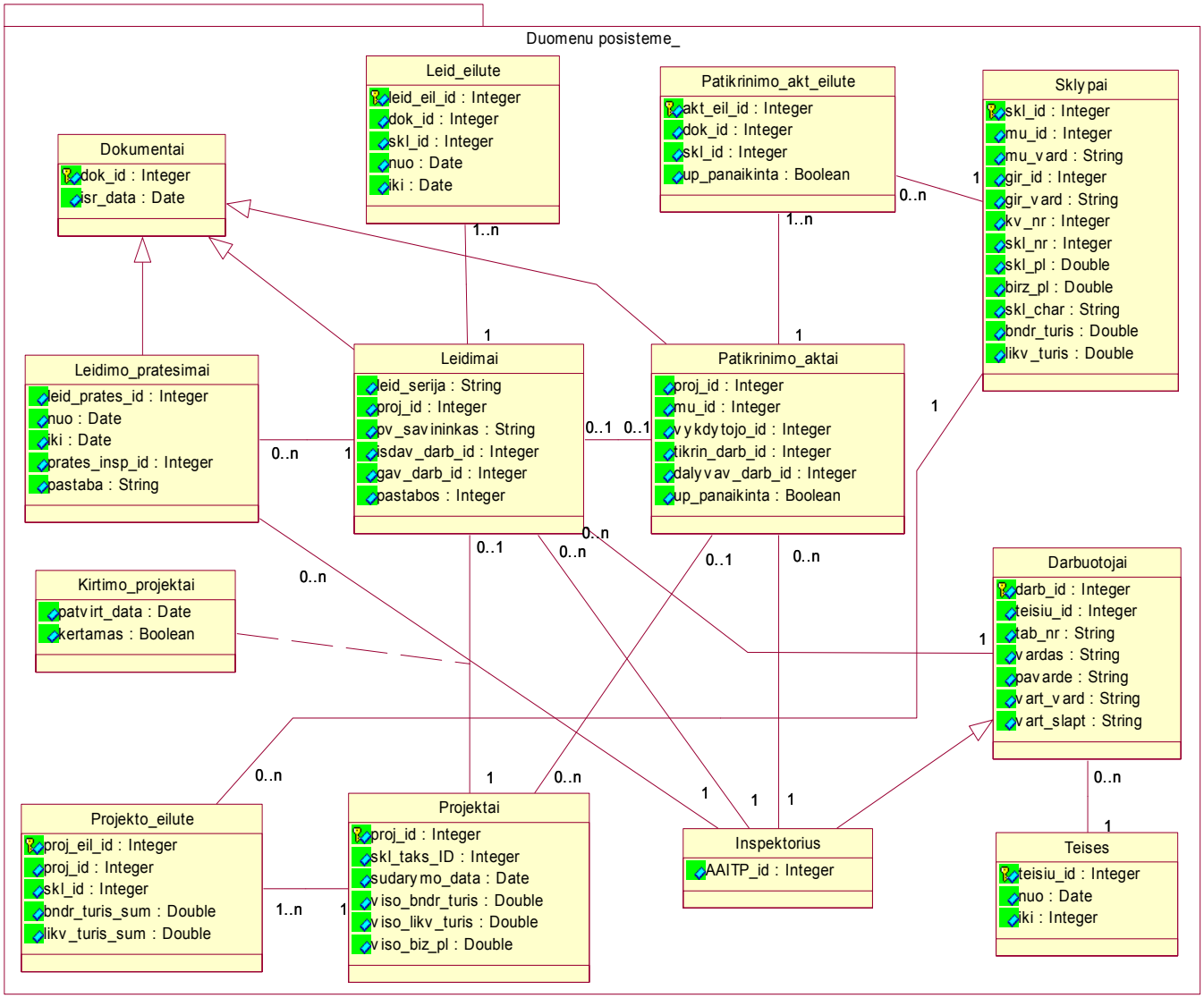
Projekto_eilute. Lentelė skirta kiekvieno projekto sklypo informacijos saugojimui.

Kirtimo_projektai. Lentelėje saugomi duomenys apie patvirtintus kirtimo projektus.

Darbuotojai. Lentelėje saugomi visų LMIIIS DB registruotų darbuotojų ir kt. su sistema susijusių asmenų (pvz., AAITP inspektorių) duomenys.

Teises. Lentelėje saugomos vartotojų teisės. Jos suteikiamos kiekvienam darbuotojui.

Sklypai. Lentelė saugoma informacija apie registruojamus sistemoje miško sklypus.

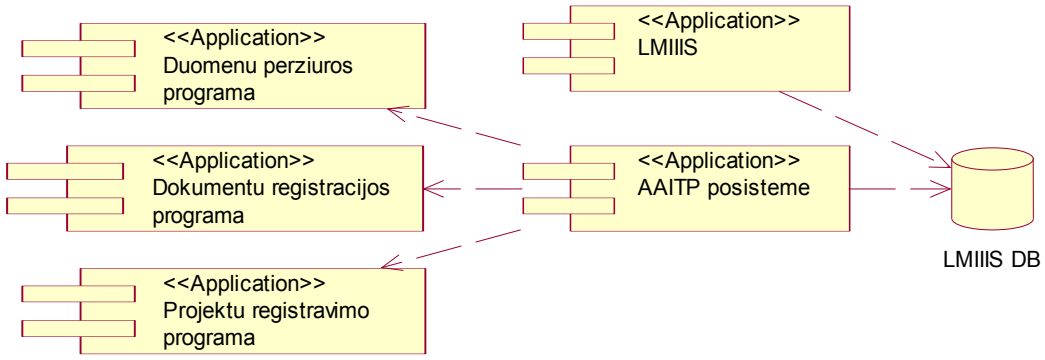


3.5 pav. Dalykinės srities klasių diagrama

4. EKSPERIMENTINĖ AAIP PROGRAMINĖS ĮRANGOS REALIZACIJA

4.1. Diegimo aplinka

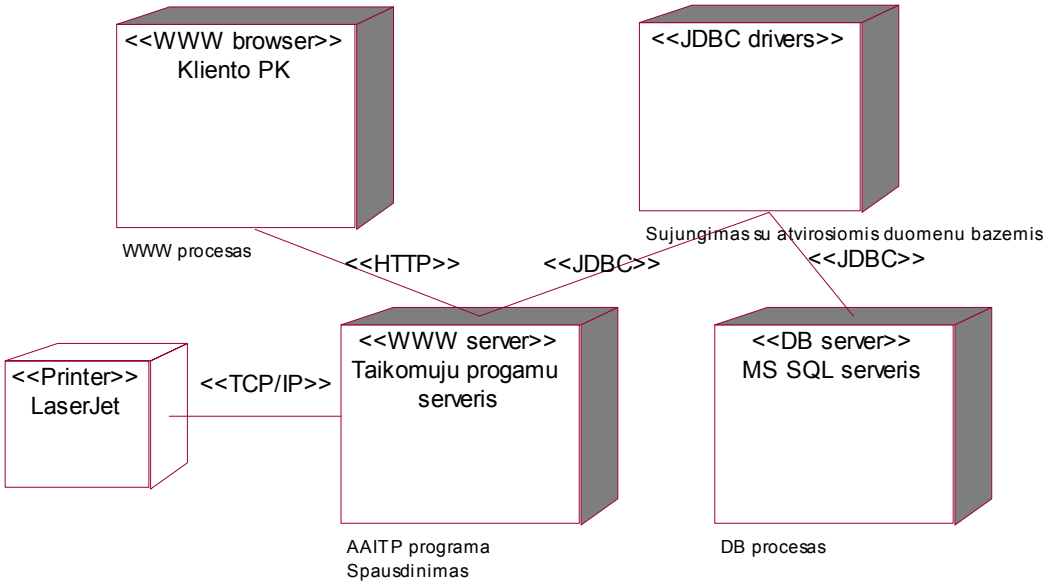
Sistemos pagrindinių komponentų pateikiama 4.1 paveiksle.



4.1 pav. Sistemos komponentų diagrama

Šiam projektui buvo pritaikytas klasikinis trijų lygių sistemos pasiskirstymo modelis. Vartotojas duomenis įveda ir rezultatus pamato interneto naršyklėje. Ši bendradarbiauja su programine įranga per programų serverį *Tomcat*, kuriame įdiegtas *Struts* karkasas, *Java Beans* komponentai ir vartotojo sąsaja (JSP). Duomenys saugojami duomenų bazėje, kitoje programinėje stotyje (4.2 pav.).

Vartotojo ir sistemos sąveika vykdoma HTTP protokolo pagalba, o programų sistema su duomenų susijungia duomenų bazių tvarkyklių pagalba.



4.2 pav. Paskirstymo diagrama

Programinės įrangos įdiegimas. Diegiant programų sistemą naujame serveryje reikia:

- Pasirūpinti, kad ten būtu įdiegtas Java SDK 1.4 arba naujesnis paketas.
- Suinstaliavus Java SDK paketą, reikia įdiegti servletų konteinerį, kuriame bus apdorojama sukurta programų sistema. Rekomenduojama naudoti Tomcat 4.x (ar vėlesnės versijos) servletų konteinerį.
- Prieš patalpinant sistemą į servletų konteinerį, reikia įsitikinti, kad programų sistema naudos tinkamą duomenų bazę duomenims saugoti.
- Sėkmingai patalpinus sistemą į servletų konteinerį, programų sistemą bus galima naudoti adresu: <http://serverio_kur_idiegtas_Tomcat_adresas/AAITP>.

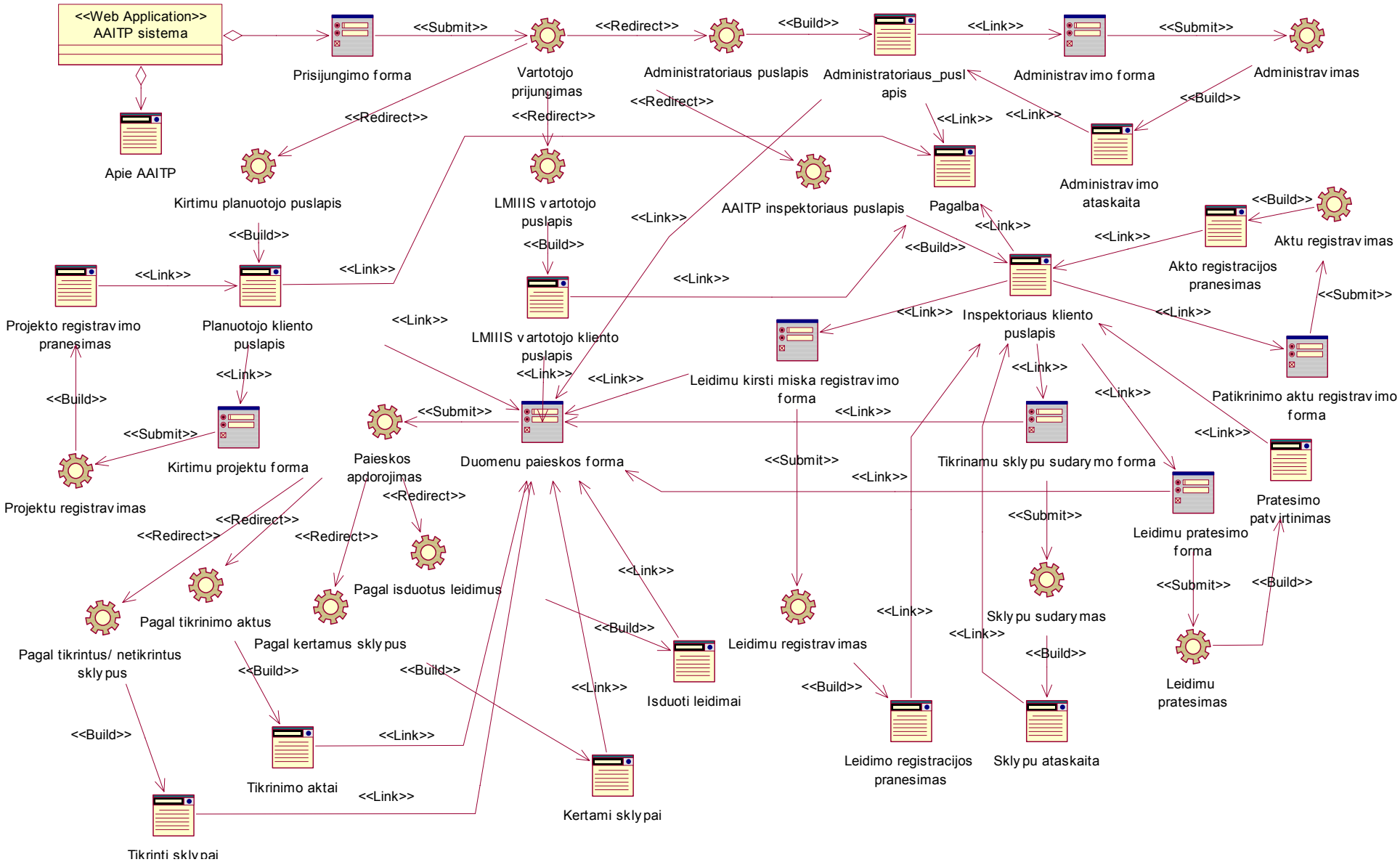
Sistemos naudojami resursai. Programinei sistemai paleisti reikalinga:

- servletų konteineris (*Tomcat*);
- MS SQL DBVS duomenims saugoti.

Šie programiniai paketai yra programinės sistemos dalys, reikalingi jos veikimui. Papildomai su niekuo nebendradarbiaujama. Sistema gali būti diegiama dviejuose serveriuose (programiniai moduliai įdiegiami ir duomenys išskiriami į atskiras tarnybines stotis) arba viename (DBVS ir servletų konteineris – tame pačiame serveryje).

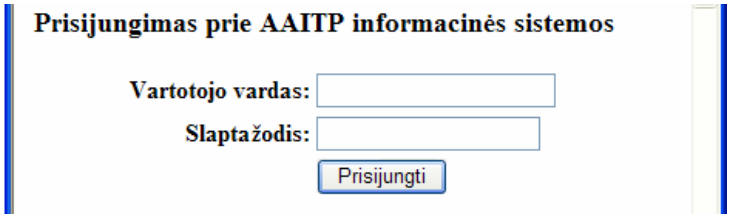
4.2. Vartotojo sąsajos realizavimas

Toliau trumpai aprašomi pagrindiniai AAIP informacijos sistemos sąsajų komponentai – *Web* puslapiai ir formos. AAIP svetainės schema pateikta 4.3 paveiksle, o toliau skyriuje pateikti realizuotos sistemos vartotojo sąsajos pavyzdžiai.



4.3 pav. Detalizuota AAITP svetainės schema

Vartotojo prieiga yra realizuojama teisių suteikimo ir tikrinimo pagrindu. Kiekvienam vartotojui, kuris yra prisiregistravęs sistemoje yra suteikiamos teisės. Vartotojui formuojant sistemos tinklalapį yra tikrinama, kokias teises jis turi ir ar visus veiksmus gali atlikti (1.1 pav.).



4.4 pav. Sistemos prisijungimo langas

Sėkmingai prisijungus, vartotojui pateikiamas pagrindinis langas su sistemos funkcijomis (4.2 pav.). Meniu suskirstytas į dalis. Jei meniu dalys yra neaktyvios, vadinasi, vartotojas neturi joms teisių.

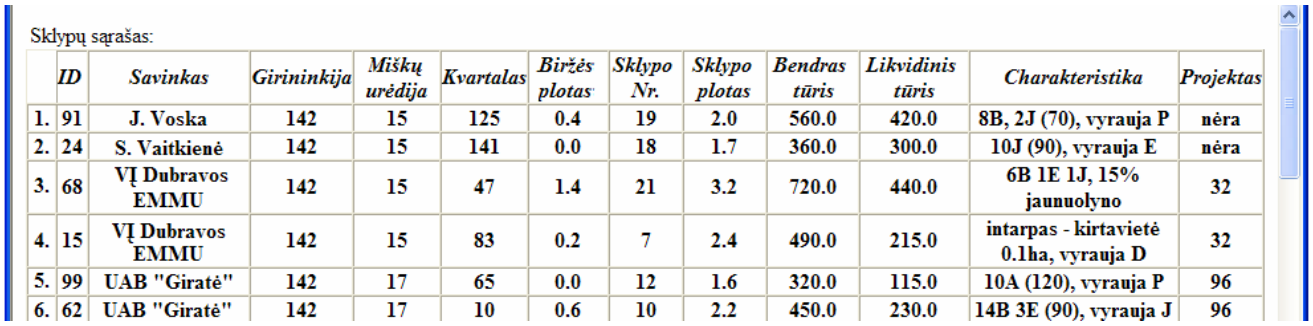
4.5 pav. Pagrindinis meniu langas

Toliau trumpai aprašytos pagrindinės sistemos funkcijos:

- Sklypų registravimas – funkcija, leidžianti užregistruoti naują sklypą sistemoje. Sėkmingai įvedus duomenis, sistema grįžta į pagrindinį meniu (4.3 pav.).

4.6 pav. Naujo sklypo registravimo forma

- Sklypų peržiūra – pateikiamas registruotų sistemoje sklypų detalus sąrašas. Prie kiekvieno sklypo yra laukas, nurodantis jo priklausomybę projektui. Jei sklypas yra įtrauktas į projektą, nurodomas projekto identifikacinis numeris sistemoje (4.4 pav.).



Sklypų sąrašas:

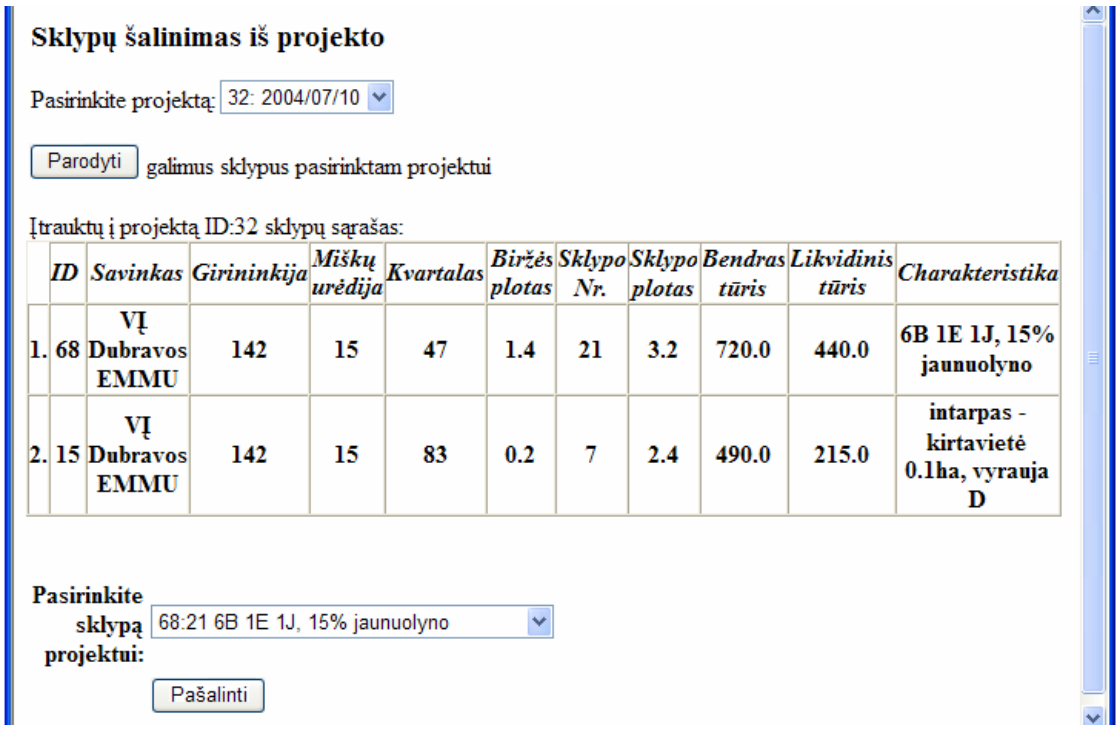
	ID	Savinkas	Girininkija	Miškų urėdija	Kvartalas	Biržės plotas	Sklypo Nr.	Sklypo plotas	Bendras tūris	Likvidinis tūris	Charakteristika	Projektas
1.	91	J. Voska	142	15	125	0.4	19	2.0	560.0	420.0	8B, 2J (70), vyrauja P	nėra
2.	24	S. Vaitkienė	142	15	141	0.0	18	1.7	360.0	300.0	10J (90), vyrauja E	nėra
3.	68	VĮ Dubravos EMMU	142	15	47	1.4	21	3.2	720.0	440.0	6B 1E 1J, 15% jaunuolyno	32
4.	15	VĮ Dubravos EMMU	142	15	83	0.2	7	2.4	490.0	215.0	intarpas - kirtavietė 0.1ha, vyrauja D	32
5.	99	UAB "Giratė"	142	17	65	0.0	12	1.6	320.0	115.0	10A (120), vyrauja P	96
6.	62	UAB "Giratė"	142	17	10	0.6	10	2.2	450.0	230.0	14B 3E (90), vyrauja J	96

4.7 pav. Sklypų peržiūros langas

- Projektų registravimas – sistemos funkcija, leidžianti įtraukti naują projektą į sistemą. Sėkmingai įvedus duomenis, sistema grįžta į pagrindinį meniu.
- Projektų peržiūros lange pateikiamas registruotų sistemoje projektų detalus sąrašas. Pateikiama informacija apie kiekvieno projekto sudarymo datą, jo bendrą bei likvidinį tūrį.
- Paskirti darbuotoją – šios funkcijos pagalba vartotojas gali projektui priskirti darbuotoją, dirbantį su projektu. Paskyrimas vykdomas trimis etapais: 1) pasirenkamas projektas, kuriam norima paskirti darbuotoją ir paspaudžiamas mygtukas Parodyti; 2) išfiltruojami duomenys; 3) pasirenkamas bei patvirtinamas darbuotojas. Vienam projektui galima paskirti tik vieną darbuotoją (4.5 pav.).

4.8 pav. Darbuotojo paskyrimo projektui langas

- Darbuotojų projekte sąrašas – pateikiama informacija apie darbuotoją, paskirtą projektui. Norint pamatyti rezultatus, pirmiausia reikia pasirinkti projektą ir paspausti mygtuką Parodyti.
- Pašalinti darbuotoją iš projekto – šios funkcijos pagalba vartotojas gali pašalinti projektui priskirtą darbuotoją. Šalinimas vykdomas adekvačiai paskyrimui: 1) pasirenkamas projektas, iš kurio norima pašalinti darbuotoją; 2) išfiltruojami duomenys; 3) pasirenkamas bei patvirtinamas šalinimui darbuotojas.
- Įtraukti sklypą į projektą – vartotojas gali projektui priskirti miško sklypą. Paskyrimas vykdomas trimis etapais: 1) pasirenkamas projektas, kuriam norima paskirti sklypą; 2) išfiltruojami duomenys; 3) pasirenkamas bei patvirtinamas sklypas.
- Sklypų projekte sąrašas – pateikiama informacija apie sklypus, įtrauktus į projektą.
- Pašalinti sklypą iš projekto – vartotojas gali pašalinti sklypą iš pasirinkto projekto. Šalinimas vykdomas adekvačiai paskyrimui: 1) pasirenkamas projektas, iš kurio norima pašalinti sklypą; 2) išfiltruojami duomenys; 3) pasirenkamas bei patvirtinamas šalinimui sklypas (4.6 pav.).



	ID	Savinkas	Girininkija	Miškų urėdija	Kvartalas	Biržės plotas	Sklypo Nr.	Sklypo plotas	Bendras tūris	Likvidinis tūris	Charakteristika
1.	68	VĮ Dubravos EMMU	142	15	47	1.4	21	3.2	720.0	440.0	6B 1E 1J, 15% jaunuolyno
2.	15	VĮ Dubravos EMMU	142	15	83	0.2	7	2.4	490.0	215.0	intarpas - kirtavietė 0.1ha, vyrauja D

4.9 pav. Sklypų šalinimo iš pasirinkto projekto sąrašas

- Vartotojų registravimas (sistemos dalis, skirta išimtinai sistemos administratoriui) – funkcija, leidžianti užregistruoti naują vartotoją sistemoje. Sėkmingai įvedus duomenis, sistema grįžta į pagrindinį meniu.
- Vartotojų peržiūros lange (sistemos dalis, skirta išimtinai sistemos administratoriui) pateikiamas visų sistemoje registruotų vartotojų sąrašas.

4.3. Sistemos kokybės analizė

Produkto veikimo korektiškumas buvo atliktas testavimo metu. Pagrindiniai realizuotos sistemos kokybės analizės tikslai:

- aptikti klaidas funkcionavime, logikoje, realizacijoje;
- patikrinti, ar programų sistema atitinka reikalavimų specifikaciją;
- įsitikinti, ar programų sistema sukurta pagal standartus.

Darbo kokybės vertinimas. Analizuojant šio projekto kokybę, buvo apklausiami prototipą naudoję KTU darbuotojai:

- doc. L. Nemuraitė (projekto vadovė);
- L. Čeponienė (Informacijos sistemų katedros doktorantė);
- M. Balandytė (Informacijos sistemų katedros darbuotoja).

Sistemos prototipą testavę vartotojai tyrimo metu užpildė 4.1 lentelę, kurioje įvertino (5 – puikiai tenkina, 4 – tenkina, 3 – priimtina, 2 – silpnai, 1 – netenkina) AAI informacijos sistemos atitikimą specifikacijai.

4.1 lentelė. Produkto vertinimo rezultatai

Nr.	Funkcionalumas	Testuotojai		
		LN	LČ	MB
1.	Prisijungimas prie darbo aplinkos	5	5	5
2.	Atsijungimas nuo darbo aplinkos	5	5	5
3.	Naujo sklypo registravimas	4	5	4,5
4.	Projektų registravimas	4	4	4
5.	Darbuotojų paskyrimas projektui	5	5	4
6.	Darbuotojų šalinimas iš projekto	5	4	4
7.	Sklypų įtraukimas į projektą	5	4,5	4
8.	Sklypų šalinimas iš projekto	5	4	4
9.	Informacijos peržiūra	4	4	5
10.	Vartotojų registravimas	4	3,5	3,5

Taip pat buvo įvertinti ir nefunkciniai reikalavimai sistemos panaudojamumui (4.2 lentelė).

4.2 lentelė. Sistemos panaudojamumo įvertinimas

Nr.	Kriterijus	Testuotojai		
		LN	LČ	MB
1.	Vartotojo sąsajos aiškumas	5	5	5
2.	Perteklinių veiksmų nebuvimas	5	4,5	5
3.	Užduočių vykdymo greitis	5	5	5
4.	Duomenų ir meniu punktų išdėstymo patogumas	3,5	4	4
5.	Pagalba vartotojui	5	4	5

Kadangi projekto realizacija buvo nutraukta metams, AAIP programinės įrangos sistemą buvo nuspręsta jį realizuoti ne kaip LMIIIS posistemę, bet kaip atskirą sistemą. Taigi sistemos prototipas ne visiškai išpildo specifikacijoje numatytas sistemos savybes. Tačiau atlikus kai kuriuos modifikavimus,

realizuotas sistemos prototipas gali nesunkiai pasiekti projekto pradžioje planuotą pilną savo funkcionalumą arba likti prototipu kitų panašaus funkcionalumo sistemų kūrimui ateityje.

Kalbant apie praplėtimo galimybes – pasirinkta architektūra leidžia nesunkiai padidinti Aplinkos apsaugos inspekcijos informacinės sistemos paslaugas, tačiau JSP vartotojo sąsaja pareikalautų daugiausiai darbo atliekant pakeitimus.

IŠVADOS

1. Darbo tikslas buvo išnagrinėti didelių sistemų projektavimo ypatumus, jų išskaidymo į posistemius metodikas, išanalizuoti metodikų privalumus bei trūkumus, o taip pat sukurti interneto prieigą Aplinkos apsaugos inspekcijos teritoriniams padaliniams pritaikant įsisavintas teorines žinias.
2. Kuriant Aplinkos apsaugos inspekcijos IS, buvo susidurta su didelių, iš posistemių sudarytų sistemų kūrimo problemomis. Norint sukurti plečiamą sistemą su pakartotino panaudojimo elementais, buvo išnagrinėti sistemų išskaidymo į posistemius metodai:
    a) struktūrinė metodika (besiremianti funkciniu komponentų išskaidymu);
    b) objektinis modeliavimas.
3. Suformuluotas posistemių projektavimo metodas, kuris pradiniam išskaidymui naudoja funkcinius, o toliau – objektinių metodų principus. Gauta posistemių architektūra užtikrina lanksčias posistemių priklausomybes ir daugkartinio naudojimo galimybes.
4. Siūlomas metodas buvo pritaikytas projektuojant Aplinkos apsaugos inspekcijos informacinę sistemą. Sistemos prototipas buvo realizuotas naudojant J2EE technologijas.
5. Sukurta programinė įranga įgyvendina reikalavimus didelių paskirstytų sistemų naudojimui ir palaikymui (didelio skaičiaus paskirstytų vartotojų prieiga, teisių apsauga, informacinės bazės suderinamumas, centralizuota sistemos priežiūra, plečiamumas).
6. Prototipą ir sudarytą metodiką galima panaudoti plečiant Aplinkos apsaugos inspekcijos IS bei kuriant kitas panašias sistemas.
7. Darbo tematika buvo publikuotas straipsnis X-oje tarpuniversitetinės magistrantų ir doktorantų konferencijos “Informacinės technologijos 2005” pranešimų medžiagos leidinyje.

LITERATŪRA

1. ČERNIAUSKAITĖ, A. *Aplinkos apsaugos inspekcijos teritorinių padalinių programinė įranga. Projekto galutinė dokumentacija* (cerniauskaite_final.doc). Magistrinis projektas, Programų inžinerijos katedra, KTU, 2004, 14–22, 35–46, 91–101 p.

2. ČERNIAUSKAITĖ, A. *Universalaus proceso taikymas posistemiams projektuoti*. X tarpuniversitetinė magistrantų ir doktorantų konferencija , Informacinės technologijos 2005“: konferencijos pranešimų medžiaga. Kaunas: Technologija, 2005, 213–216 p.

3. CLOUTIER, R., WINKLER, A., WATSON, J., FICKLE, C. *Modeling a System of Systems Using UML* [interaktyvus]. The 3rd Annual Conference on Systems Engineering Research, Stevens Institute of Technology, 2005 04 [žiūrėta 2005 m. balandžio 25 d.]. Prieiga per internetą: <http://www.calimar.com/Papers/Modeling a System of Systems using UML.pdf>.

4. EELES, P., ERICSSON, M. *Modeling for enterprise initiatives with the IBM Rational Unified Process. Part I: RUP and the System of Interconnected Systems Pattern* [interaktyvus]. The Rational Edge, Rational Software Corporation, 2003 07 [žiūrėta 2005 m. kovo 18 d.]. Prieiga per internetą: <http://www-106.ibm.com/developerworks/rational/library/content/03July/2500/2817/2817_peme.pdf>.

5. EELES, P., ERICSSON, M. *Modeling for enterprise initiatives with the IBM Rational Unified Process. Part II: Example* [interaktyvus]. The Rational Edge, Rational Software Corporation, 2003 10 [žiūrėta 2005 m. kovo 18 d.]. Prieiga per internetą: <http://www-128.ibm.com/ developerworks/rational/library/content/RationalEdge/oct03/t_modeling_pe_me.pdf>.

6. LARMAN, C. *Issues in System Design*. *Applying UML and Patterns*. Prentice Hall Inc., 1997, 271, 275–279 p.

7. LYKINS, H., FRIEDENTHAL, S., MEILICH, A. *Adapting UML for an Object Oriented Systems Engineering Method (OOSEM)* [interaktyvus]. Process Integration for 2000 and Beyond: Systems Engineering and Software Symposium. New Orleans LA: Lockheed Martin Corporation, 2001 01 15 [žiūrėta 2005 m. balandžio 26 d.]. Prieiga per internetą: <http://www.incose.org/chesapek/meetings/Adapting_UML_for_an_OOSEM.doc>.

8. MURRAY, C. *Thoughts on Functional Decomposition* [interaktyvus]. The Rational Edge, Rational Software Corporation, 2003 04 [žiūrėta 2005 m. balandžio 26 d.]. Prieiga per internetą: <http://www-106.ibm.com/developerworks/rational/library/content/RationalEdge/apr03/ FunctionalDecomposition_TheRationalEdge_Apr2003.pdf>.

9. МУХАМЕДЗЯНОВ, Р. Р. *Серверные приложения на языке Java*. Москва: Солон-Р, 2002, с 197–205.

10. *ServerSideScripting techniques analysis* [interaktyvus, žiūrėta 2005 m. kovo 20 d.]. Prieiga per internetą: <http://www.b2bsim.de/documents/wewior/main.html>.

11. *The Java Tutorial* [interaktyvus]. Sun Microsystems, Inc. 1994–2005 [žiūrėta 2005 m. kovo 20 d.]. Prieiga per internetą: <http://java.sun.com/docs/books/tutorial/index.html>.

12. *UML. Notation Guide* [interaktyvus]. Rational Software Corporation, 1997 [žiūrėta 2005 m. balandžio 25 d.]. Prieiga per internetą: <http://www-306.ibm.com/software/rational/uml/>.

TERMINŲ IR SANTRUMPŲ ŽODYNAS

AAIP – Aplinkos apsaugos inspekcijos (teritorinių padalinių) posistemė

API – *Application Programming Interface* – standartizuotas klasių bibliotekų rinkinys programavimui Java palengvinti

CASE – *Computer Aided-Software Engineering* – programinė įranga arba programų paketai, skirti supaprastinti programų sistemų kūrimą ir palaikymą

DB – duomenų bazė

LMIIIS – Lietuvos miškų išteklių integruota informacinė sistema

LVMI – Lietuvos valstybinis miškų institutas

EJB – *Enterprise JavaBean* – SUN kompanijos paskirstytų sistemų kūrimo technologija, pagrįsta konteinerio ir integruotų elementų koncepcija. EJB konteineris – speciali sisteminė aplinka, aprašanti visų EJB komponentų darbą (turi būti serveryje, palaikančiame J2EE platformą)

HTTP – *Hypertext Transfer Protocol* – protokolas, skirtas duomenų perdavimui pasauliniame tinkle

IDE – *Integrated Development Environment* – integruota kūrimo aplinka, t. y. programos turinčios priemones (automatinio teksto pabaigimo ir jo generavimo įrankiai, kompiliatoriai, grafinė vartotojo sąsaja ir pan.), palengvinančias bei pagreitinančias programinės įrangos kūrimo procesą

Java – objektinė programavimo kalba

JDBC – *Java DataBase Connectivity* – sujungimo su DB naudojant Java technologija

JDK – *Java Developer's Kit* – programų paketas su API rinkiniu bei pagrindiniu įrankių rinkiniu Java programų kūrimui

J2EE – *Java 2, Enterprise Edition* – kūrimo Java platforma

JSP – *Java Server Pages* – dinaminiai puslapiai, integruojantys Java kodą

PA – panaudojimo atvejis (sistemos elgsenos vienetas – vartotojo ir sistemos sąveikų seka, kuri duoda vartotojui reikšmingą rezultatą)

PHP – *Hypertext Pre-Processor* – interpretuojama programavimo kalba, specialiai pritaikyta internetinių svetainių kūrimui

SQL – *Structured Query Language* – struktūrizuotų užklausų kalba

TCP/IP – *Transmission Control Protocol/Internet Protocol* – standartinis interneto protokolas duomenų perdavimui

UML – *Unified Modelling Language* – veiklos ir programų sistemų projektavimo rezultatų vaizdavimo, specifikavimo, konstravimo ir dokumentavimo kalba

Administratorius – LMIIIS vartotojas, turintis visas ar dalines teises keisti sistemos konfigūraciją ir informaciją. Tokias pačias teises jis turės ir AAIP.

Aktorius – subjektas (asmuo, organizacinis vienetas, sistema), kuris sąveikauja su kuriama sistema.

Girininkija – miškų urėdijos padalinys, kur vykdoma miškų ūkio veikla.

Inspektorius – aplinkos apsaugos inspekcijos arba jos teritorinio padalinio darbuotojas, šiame projekte priklausantis didžiausią prioritetą turinčių vartotojų kategorijai.

Kirtimų planuotojas – darbuotojas, taksuojantis mišką, t. y. įvertinantis miško sklypus, sudarantis kirtimo projektus bei registruojantis šią informaciją duomenų bazėje.

Leidimas – formalus dokumentas, išduodamas AAIP inspektoriaus ir leidžiantis projekto vykdytojui (girininkijai, urėdijai, miško sklypo savininkui ar jo įgaliotam atstovui) iškirsti kirtimo projekte numatytus miško sklypus.

Miškų urėdija – valstybės įmonė, turto patikėjimo teise valdanti, naudojanti valstybinius miškus ir jais disponuojanti įstatymų nustatyta tvarka, taip pat vykdanti juose kompleksinę miškų ūkio veiklą ir kitą įmonės nuostatuose numatytą veiklą.

Patikrinimo aktas – privačių valdų individualių miškotvarkos projektų patikrinimo aktas – formalus dokumentas, surašomas patikrinus projekte įtrauktus miško sklypus.

Projektas – sklypų kirtimo projektas, sudarytas taksatoriaus ir patvirtintas komisijos.

Sklypas – miško sklypas, kurio taksacinės charakteristikos duomenys užregistruoti LMIIS duomenų bazėje.

1 PRIEDAS. Konferencijos straipsnis

UNIVERSALAUS PROCESO TAIKYMAS POSISTEMIAMS PROJEKTUOTI

10-osios tarpuniversitetinės magistrantų ir doktorantų konferencijos „Informacinės technologijos“pranešimų leidinys.

Agnė Černiauskaitė

Kauno Technologijos Universitetas, Studentų g. 50, LT - 51368 Kaunas

Didelės sistemos dažniausiai susideda iš keleto posistemių, kurių kiekviena yra atskira taikomoji programa ar modulis, susiję tarpusavio ryšiais. Sprendžiant verslo kompiuterizavimo problemas, neretai prireikia platesnės perspektyvos, kuri apžvelgtų sistemą kaip kelių mažesnių sistemų visumą. Straipsnyje aptariamas universalaus proceso taikymas tokių sistemų projektavimui. Pasiūlyta posistemių išskyrimo metodika pritaikyta kuriant eksperimentinę Aplinkos apsaugos inspekcijos teritorinių padalinių informacinę sistemą.

1. Įvadas

Vis didėjanti šiandienos sistemų apimtis bei augantis jų sudėtingumas verčia galvoti apie sistemų sistemos projektavimą. Didelės sistemos prigimtinis sudėtingumas reikalauja tokio kūrimo proceso, kuris skaidytų sistemą į posistemius, atitinkančius tam tikrą kiekį projektų. Straipsnyje apžvelgiamos tokio sistemų projektavimo metodikos taikant universalų procesą: funkcinis sistemos išskaidymas ir objektinės orientacijos sistemų inžinerija. Pavyzdžiai straipsnyje pateikiami UML diagramų pagrindu [5], remiantis eksperimentinės Lietuvos miškų instituto informacinės sistemos Aplinkos apsaugos inspekcijos teritorinių padalinių posistemio projektu.

2. Didelių sistemų projektavimo ypatybės

Didelių sistemų projektavimui būdinga „visumos“ perspektyva, kuomet projektuojant sistemą koncentruojamasi tik į architektūros požiūriu būtinus elementus [4]. Tačiau kiekvienas posistemis privalo atspindėti detales, nes jis įgyvendina pagrindinės sistemos aspektus. Galima sakyti, kad iš posistemių sudarytos sistemos kūrimo procesas remiasi „iš viršaus žemyn“ (pagrindinė sistema suteikia turinį kiekvienam posistemiui) ir „iš apačios į viršų“ principais (kiekvienas posistemis realizuoja tam tikrą sistemos aspektą).

Vadinasi, pagrindinė sistema taip pat yra priklausoma nuo kiekvieno ją išpildančio posistemio. Taigi kas pirmiau – sistema ar posistemis? Visumos negalima apibrėžti be jos dalių techninių smulkmenų, o dalių negalima apibrėžti nesuprantant visumos. Tai parodo sistemos bei posistemių tarpusavio sąryšį. Todėl jų kūrimas turėtų būti vykdomas lygiagrečiai. Vis tik dažniau vyrauja tendencija spręsti sudėtingas problemas „iš viršaus žemyn“, kadangi didelę problemą lengviau spręsti suskaidžius ją į aibę mažesnių ir tuo pačiu lengviau išsprendžiamų. Pavyzdžiui, 1 paveiksle pateiktas Lietuvos miškų informacinės sistemos Aplinkos apsaugos inspekcijos teritorinių padalinių dalies panaudojimo atvejų modelis. Kiekvienas posistemis turi savo aktorius ir panaudojimo atvejus, apibrėžiančius posistemio reikalavimus ir kartu atitinkančius bendros sistemos reikalavimus.

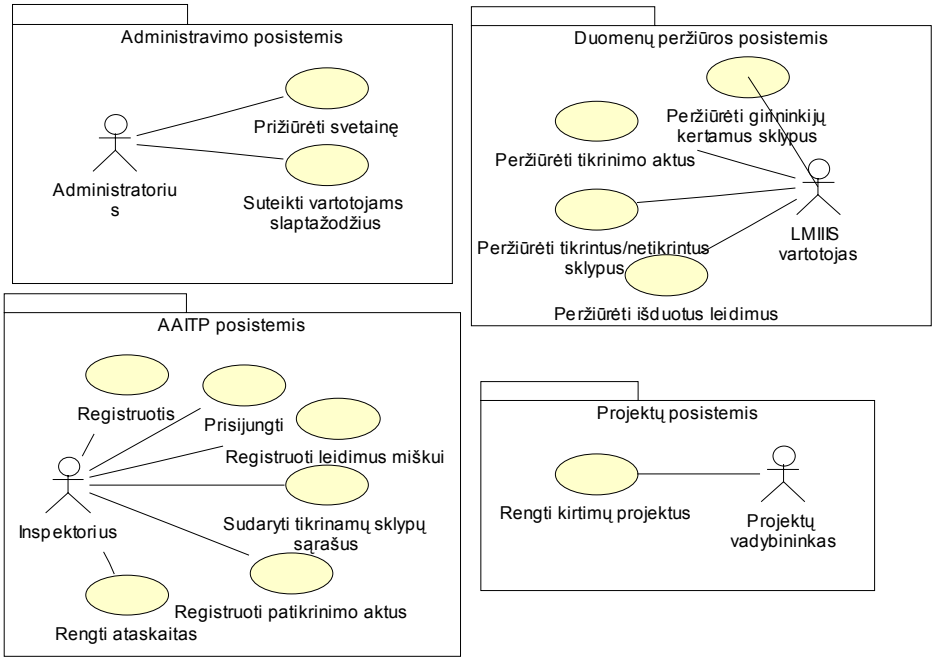
Didelių sistemų kūrime yra rizika, jog galima pamiršti atsižvelgti į tai, kaip suskaidytų problemų sprendimai įtakoja bendrą sprendimą. Susitelkus į mažų dalių kūrimą ir nuolat nesiejant jų su visuma, kūrimo pabaigoje gali tapti neįmanoma sujungti jas į bendrą visumą. Kita problema – turint daug „mažų dalių“ir norint jas sujungti (pavyzdžiui, integruojant senas sistemas), reikia apriboti sistemą atsižvelgiant tai, ką galima padaryti iš tų mažų dalių.

Tradicinis sistemų inžinerijos metodas modeliuoti sudėtingas sistemas – atlikti funkcinį išskaidymą tam, kad būtų galima nustatyti pagrindines sistemos funkcijas ir, tai įvykdžius, pritaikyti tą pačią metodiką didesnių funkcijų suskaidymui į mažesnes. Šis metodas nukreipia sistemos apibrėžimą labiau inžinerinės, o ne vartotojo perspektyvos link ir gali sąlygoti technologiškai sudėtingas, nebūtinai vartotojo poreikius atitinkančias sistemas.

Antras didelių sistemų projektavimo metodas – objektinių sistemų inžinerija. Pastaroji koncentruojasi ties iš posistemių sudarytos sistemos specifikavimo ir skaidymo problemomis. Objektinė metodologija [1] apibrėžia tai, ko vartotojas tikisi iš sistemos (kaip ji veiks daugelyje skirtingų situacijų), ir sukuria panaudojimo atvejų modelį, apimantį šiuos scenarijus, bei klasių modelį, valdantį informaciją ir funkcionalumą. Šie du UML modeliai suteikia pakankamai informacijos sukurti sistemą, suskaidytą į posistemius, kurių kiekvienas turi savo panaudojimo atvejus ir klases. Tačiau nėra tiksliai apibrėžtos metodikos, kaip tuos posistemius išskirti. Universaliame projektavimo procese pateikti keli apibendrinti posistemių struktūros variantai (funkcinė; trijų lygių architektūra, išskiriant

sąsajos, veiklos paslaugų ir duomenų paslaugų posistemius; daugiasluoksnė), tačiau detalesnių rekomendacijų, kaip išskirti pakartotino naudojimo posistemius, nėra. Tai paliekama projektuotojo intuicijai.

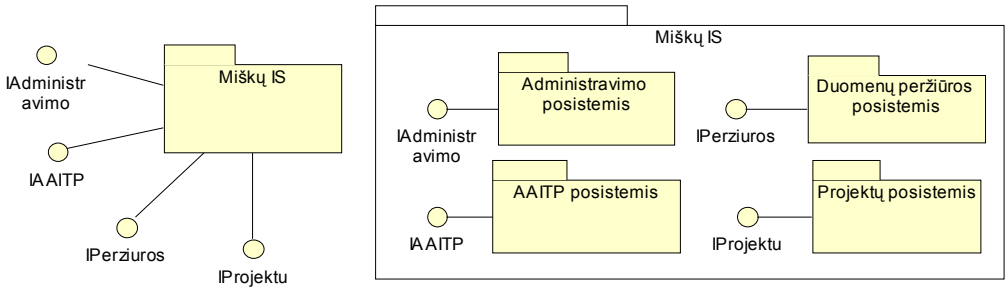
Šiame straipsnyje siūloma išskaidyti sistemą į posistemius iteraciniu būdu, nagrinėjant jos panaudojimo atvejų specifikacijas. Išskaidymas remiasi panašių bei pasikartojančių elementų radimu pradiniuose sistemos panaudojimo atvejų aprašymuose. Tokiu būdu išskirti posistemių panaudojimo atvejai padeda išpildyti bendros sistemos panaudojimo atvejus. Siūlomas metodas papildo [2], [3] aprašytą metodą, kur akcentuojamas atitikimas tarp sistemos ir posistemių panaudojimo atvejų.



1 pav. Lietuvos miškų informacinės sistemos dalies panaudojimo atvejai

3. Pradinis sistemos išskaidymas į posistemius

Projektuojant sistemų sistemą ir sumodeliavus projekto detales, kuriami modeliai. Svarbu suvokti modelių paskirtį ir skirtumus tarp jų. Veiklos modelis yra aukščiausio lygio, apibrėžia bendrą sistemos kontekstą. Žemesnio lygmens modelis apbrėžia pačią sistemą – jos bendrą informaciją ir funkcionalumą. Ateityje jis padės projektuotojui apibrėžti ir posistemių funkcijas. Galiausiai išskiriami sistemos posistemiai. Pradžioje pagrindinės sistemos projekto modelyje visi posistemiai laikomi „juodosiomis dėžėmis“ su kruopščiai nustatytomis tarpusavio sąsajomis (2 pav.). Vėliau kiekvienas jų suprojektuojamas atskirai, jam priskiriami komponentai, klasės ir t. t. Svarbu pažymėti, jog priklausomai nuo sistemos dydžio ir sudėtingumo, posistemiai gali būti išskaidyti į smulkesnes dalis, kurios savo ruožtu, taip pat dar gali būti skaidomos. Tokio skaidymo tikslas – valdomas sistemos apibrėžimas, kuris leistų aiškiai suprojektuoti bendrą sistemą.

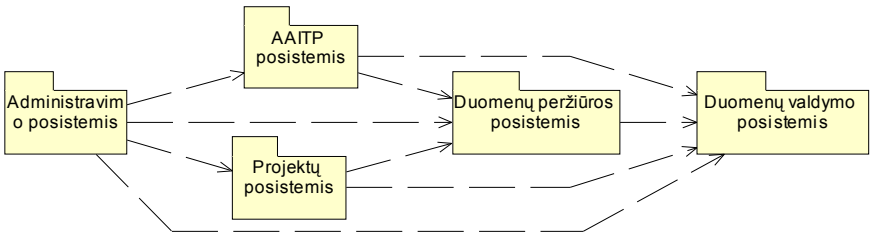


2 pav. Pradinis posistemių vaizdas

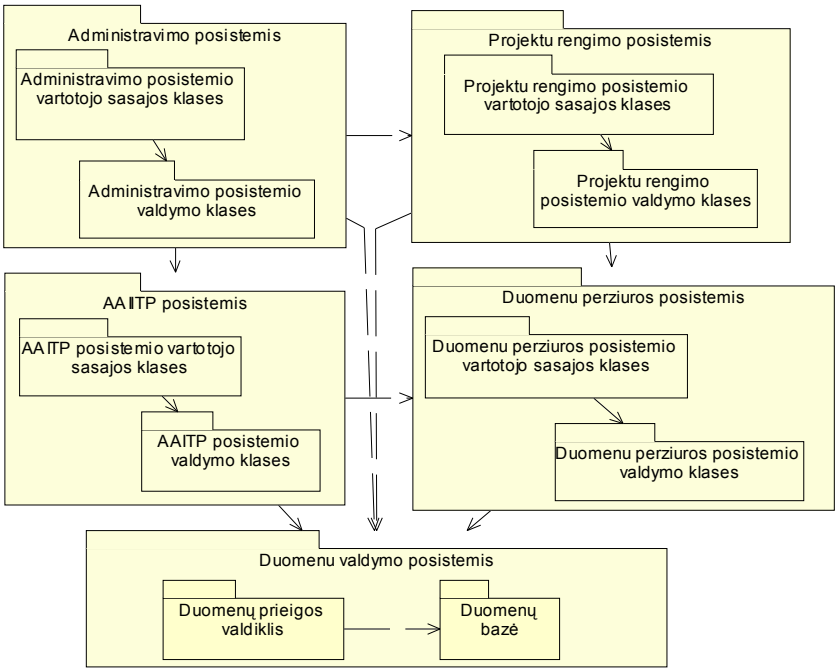
4. Posistemių identifikavimas, analizuojant panaudojimo atvejų scenarijus

Objektiniame modeliavime posistemių išskyrimas remiasi tokiais kriterijais: maksimalus vidinis posistemio glaudumas ir minimalus jo susietumas su išore. Tačiau literatūroje nėra nurodyta, kaip praktiškai išskirti posistemius pagal šiuos kriterijus. Dažniausiai projektuotojai remiasi intuicija. Paprasčiausias sistemos išskaidymas yra jos padalinimas pagal funkcines veiklos sritis (1 pav.), tačiau toks išskaidymas gali būti neefektyvus, jei posistemiuose yra pasikartojančios elgsenos grupių.

Analizuojant posistemių išskyrimo pavyzdžius, buvo pastebėta, kad posistemius galima išskirti, analizuojant sistemos elgseną – pavyzdžiui, klasių modelyje rasti pasikartojančias klasių priklausomybių sekas arba operacijų naudojimo scenarijus sekų diagramose. Dažnai tos klasės būna pavadintos skirtingais, tačiau gana panašiais vardais, ir jų skaičių galima sumažinti, įvedant apibendrinančias klases, kurias galima sugrupuoti į posistemius. Tačiau projektavimo veiklos efektyvumo požiūriu geriau tai padaryti ankstesniame sistemos projektavimo etape, analizuojant panaudojimo atvejų žingsnius. Nagrinėjamame pavyzdyje galima pastebėti, kad daugelyje panaudojimo atvejų kartojasi panašūs žingsniai, susiję su duomenų pateikimu (peržiūra) bei sąveika su duomenų baze. Administravimo, inspekcijos ir projektų rengimo posistemių panaudojimo atvejai turi savyje atitinkamų duomenų peržiūros, skaitymo, saugojimo, modifikavimo žingsnius; duomenų peržiūros panaudojimo atvejai turi savyje sąveikos su duomenų baze žingsnius. Pakartotino panaudojimo požiūriu tikslinga išskirti šiuos žingsnius į atskirus panaudojimo atvejus, kurie įeina į kitų panaudojimo atvejų sudėtį. Posistemių vaizdas po architektūros analizės pateiktas 3 paveiksle, kur atsirado du nauji posistemiai bei posistemių tarpusavio ryšiai. Detalesnis posistemių vaizdas pateiktas 4 paveiksle.



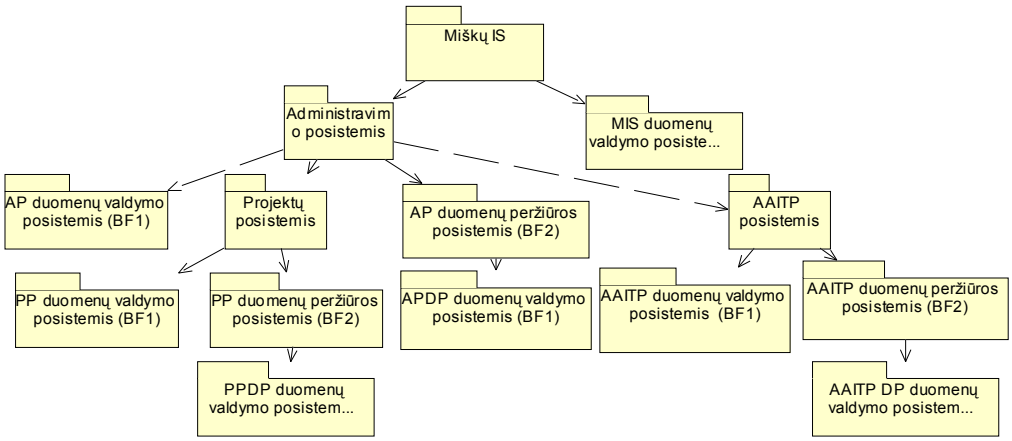
3 pav. Posistemių vaizdas po architektūros projektavimo



4 pav. Detalesnis posistemių vaizdas

Realizacijos etape posistemiai virsta programiniais komponentais. Reikia pastebėti, kad posistemių samprata objektiniame projektavime skiriasi nuo struktūriniuose metoduose naudojamos sampratos, kur sistemos posistemiai ir komponentai susideda iš žemesnio lygmens posistemių ar komponentų. Objektinėje metodologijoje vietoje agregavimo ryšio yra priklausomybė, arba naudojimo ryšys (kartais jis žymimas stereotipu <<use>>): aukštesnio lygmens komponentai naudoja žemesnio lygmens komponentus; konkretus komponentas realizuojamas ir instaliuojamas vieną kartą, o naudojamas daugelį kartų, todėl supaprastėja jo priežiūros, keitimo, tobulinimo procesai.

Struktūrinėje metodologijoje taip pat identifikuojamos bendros funkcijos (5 pav., BF1 ir BF2), kurios realizuojamos panašiais algoritmais, tačiau jos kietai susiejamos su viršesniojo lygmens komponentais, o tai labai padidina komponentų priežiūros apimtį. Projektuojant objektinėmis technologijomis realizuojamas sistemas, nepatyręs projektuotojas ar programuotojas nėra apsaugotas nuo neefektyvių panašaus pobūdžio sprendimų. Siekiant jų išvengti, siūloma atlikti panaudojimo atvejų analizę pasikartojančių scenarijų atžvilgiu ir gauti lanksčią, pokyčių atžvilgiu efektyvią architektūrą, kuri leidžia sumažinti projektavimo, kūrimo bei sistemos priežiūros darbų apimtis.



5 pav. Nagrinėjamos sistemos architektūra taikant funkcinį požiūrį

5. Išvados

Didelės sistemos dažniausiai susideda iš keleto posistemių, kurių kiekviena yra atskira taikomoji programa ar modulis, susiję tarpusavio ryšiais. Sprendžiant verslo kompiuterizavimo problemas, neretai prireikia platesnės perspektyvos, kuri apžvelgtų sistemą kaip kelių mažesnių sistemų visumą. Straipsnyje apžvelgti didelių sistemų projektavimo metodai taikant universalų procesą: funkcinis sistemos išskaidymas ir objektinis modeliavimas.

Pirmasis metodas yra paprastesnis, tačiau jo rezultate gaunama sudėtingesnė sistema, kuri gali ir nevisai atitikti vartotojo poreikius. Be to, bendros funkcijos kietai susiejamos su viršesniojo lygmens komponentais, o tai labai padidina komponentų priežiūros apimtį.

Objektiniame modeliavime posistemių išskyrimas remiasi maksimalaus vidinio posistemio glaudumo ir minimalaus jo susietumo su išore kriterijais. Metodas lankstesnis ir lengviau palaikomas, nes nėra pasikartojančių elementų. Tačiau bet kokiu atveju posistemių identifikavimo metodika yra iteracinė: sistemos skaidymas į smulkesnes dalis atliekamas tol, kol gaunamas valdomas sistemos apibrėžimas, leidžiantis aiškiai suprojektuoti bendrą sistemą.

Abu metodai buvo išbandyti, projektuojant Aplinkos apsaugos inspekcijos teritorinių padalinių informacinę sistemą. Kadangi objektinis skaidymo būdas rezultate davė ryškiai pranašesnę posistemių struktūrą, sistemos prototipas buvo realizuotas naudojant objektinį posistemių modelį ir Java technologijas. Galima teigti, kad funkcinis skaidymas gali būti naudojamas pradiniam išskaidymui; detalizuojant posistemių struktūrą tikslinga pereiti prie siūlomo skaidymo būdo, labiau priimtino objektinei metodologijai.

Literatūros sąrašas

[1] **C. Larman**. Issues in System Design. *Applying UML and Patterns*, *Prentice Hall Int*, 1997, 271, 275–279 p.

[2] **P. Eeles, M. Ericsson**. Modeling for enterprise initiatives with the IBM Rational Unified Process. Part I: RUP and the System of Interconnected Systems Pattern. *The Rational Edge*, *Rational Software Corporation*, 2003 07, 1–10 p.

[3] **P. Eeles, M. Ericsson**. Modeling for enterprise initiatives with the IBM Rational Unified Process. Part II: Example*. The Rational Edge*, *Rational Software Corporation*, 2003 07, 1–14 p.

[4] **R. Cloutier, A. Winkler, J. Watson, C. Fickle**. Modeling a System of Systems Using UML. *The 3rd Annual Conference on Systems Engineering Research, Stevens Institute of Technology*, 2005 04 23, 1–7 p.

[5] UML. Notation Guide. *www.rational.com/uml, Rational Software Corporation*, 1997.

Using the Unified Process for Subsystems Design

Large systems usually consist of a few subsystems of which each is a single interconnected software application or module. Solving business problems typically requires a broader perspective that views the system as consisting of other systems. The article looks at how Unified Process can be applied in this broader context. Suggested system decomposition method has been applied for the development of experimental Information System of Territorial Departments of Environmental Inspection.