ŠIAULIŲ UNIVERSITETAS
MATEMATIKOS IR INFORMATIKOS FAKULTETAS
INFORMATIKOS KATEDRA

AURELIJUS BRUŽAS
Informatikos specialybės II kurso neakivaizdinio skyriaus studentas

WEB APLIKACIJŲ PROGRAMAVIMO KALBA

MAGISTRO DARBAS

Darbo vadovas
Prof. habil. dr. G. Kulvietis

Recenzentas
Lekt. L. Kaklauskas

Šiauliai, 2006/2007 m.m.

Paveikslų sąrašas

Turinys

Skyrius 1

Įvadas

1.1. Temos aktualumas

1.1.1. Naujų programavimo priemonių poreikis

Augant informacinių technologijų taikymui tiek jau tapusiose tradicinėmis, tiek naujose sferose, auga ir naujos specializuotos programinės įrangos poreikis. Šiam poreikiui patenkinti dažniausiai naudojamos egzistuojančios taikomosios programos, arba diegiami ir adaptuojami specializuoti paketai. Tačiau pasitaiko atvejų, kai egzistuojančiomis priemonėmis pasiekti norimo rezultato neįmanoma, arba laiko ir kaštų sąnaudos būna mažesnės kuriant naują programinį produktą, kuris bus naudojamas tik toje srityje, ar konkretaus vartotojo.

Naujų programų gausa ir sudėtingumas skatina ieškoti būdų kaip pagreitinti ir supaprastinti jų kūrimą, užtikrinti lengvą priežiūrą ir pritaikymą prie nuolat besikeičiančių sąlygų, bei pagerinti daugelį kitų rodiklių. Tam tikslui kuriami nauji programavimo įrankiai ir metodai, savo apimtimi kintantys nuo dažniausiai pasitaikančių uždavinių sprendimo automatizavimo, iki paties programavimo sampratos keitimo. Daugelio šiuolaikinių programų kūrimas be tokių įrankių būtų komplikuotas ar neįmanomas, todėl, galima teigti, kad nauji programavimo įrankiai iš dalies skatina ir pačių jais kuriamų programų vystymąsi.

1.1.2. WEB aplikacijų plitimas

Vis daugiau naujų programų kuriama WEB pagrindu – t.y. taikant kliento/serverio tipo architektūrą, kai klientui naudojama WEB naršyklė, o pati programa veikia nutolusiame kompiuteryje. Tokio tipo aplikacijos turi keletą svarių privalumų: nereikalauja diegimo ir atnaujinimų vartotojo kompiuteryje, yra nepriklausomos nuo vartotojo kompiuterio platformos, nereikalauja didelių vartotojo kompiuterio resursų ir t.t. Pagrindinis WEB aplikacijų trūkumas – vartotojo sąsajos ribotumas. Tačiau šis trūkumas dalinai pašalinamas naudojant JavaScript, AJAX, Flash, VRML, bei kitas technologijas. Atsižvelgiant į tai, kad dauguma programų nekelia ypatingų reikalavimų vartotojo sąsajai, tikėtina, kad ateityje WEB aplikacijos dominuos [1].

WEB – palyginus, nauja aplikacijų terpė (manoma, kad pirmoji WEB aplikacija buvo parašyta 1995 metais), todėl gerai išvystytų ir patikrintų WEB aplikacijų kūrimo metodų nėra daug. Kita vertus, nuolat kuriami nauji įrankiai, leidžiantys greitai ir paprastai kurti WEB aplikacijas, dažniausiai paremti MVC architektūriniu šablonu [2] (populiariausi – Ruby On Rails, Django, ir t.t.), kurių teikiamų galimybių daugumai aplikacijų pakanka. Tačiau, siekiant išnaudoti WEB terpės potencialą, būtina skatinti WEB aplikacijų vystymąsi, bei ieškoti kokybiškai naujų programavimo būdų.

1.2. Darbo tikslai

Programavimo kalba yra įrankis, nuo kurio tinkamumo konkrečiai užduočiai labai priklauso tiek programavimo procesas (greitis, paprastumas), tiek gautas rezultatas – programa, todėl buvo pasirinkta kurti WEB aplikacijoms skirtą programavimo kalbą, ir iškelti šie darbo tikslai:

- Pirminis tikslas – sukurti WEB aplikacijų programavimo kalbą, kuri leistų greitai ir paprastai kurti įvairaus dydžio WEB aplikacijas. Dažnas greito aplikacijų kūrimo sistemų trūkumas, labiausiai lemiantis jų nepopuliarumą, – komplikuota programų priežiūra ir vystymas, todėl pagrindinis reikalavimas, keliamas kuriamajai kalbai, – išvengti šio trūkumo.

- Antrinis tikslas – sukurti ir realizuoti programinius įrankius, leidžiančius kurti WEB aplikacijas naudojant sukurtąją kalbą. Šis tikslas svarbus ne tik praktiniu, bet ir tiriamuoju požiūriu, kadangi leis realiais pavyzdžiais įsitikinti kalbos veiksmingumu, išskirti trūkumus ir privalumus. Savaime suprantama, šių įrankių realizacija viršytų šio darbo apimtį, todėl bus kuriamas supaprastintas variantas, kurio galimybių pakaktų minimalių WEB aplikacijų programavimui.

1.3. Darbo uždaviniai

Siekiant pagrindinio darbo tikslo, turi būti įgyvendinti šie uždaviniai:

- Išskirtos pagrindinės egzistuojančios WEB aplikacijų kūrimo priemonės ir metodai, išskirti pagrindiniai šių priemonių ir metodų privalumai ir trūkumai, bei nustatytos jų priežastys;

- Išanalizuota keletas tipinių WEB aplikacijų, išskiriant pagrindines užduotis, veikimo principus, struktūrą.

- Suformuoti reikalavimai kuriamajai kalbai, atsižvelgiant į užsibrėžtus tikslus, tipinių WEB aplikacijų poreikius, bei išnagrinėtų priemonių privalumus ir trūkumus;

- Sukurta ir aprašyta WEB aplikacijų programavimo kalba, tenkinanti kiek įmanoma daugiau iškeltų reikalavimų.

Siekiant antrinio tikslo, turi būti įgyvendinti šie uždaviniai:

- Sukurti ir aprašyti formatai, kuriais bus saugomos kuriamąja kalba parašytos programos (reikalinga tuo atveju, jei programos leidžiamos naudojant interpretatorių);

- Suprojektuota ir realizuota programa, leidžianti šias programas įvykdyti kompiuteryje (kompiliatorius arba interpretatorius);

- Suprojektuota ir realizuota programa, leidžianti užrašyti šias programas.

Taip pat, siekiant patikrinti sukurtos kalbos tinkamumą ir palyginti su kitomis, turėtų būti sukurta tipinė WEB aplikacija, ir realizuota naująja, bei kuria nors egzistuojančia priemone.

1.4. Mokslinis naujumas

Nors WEB aplikacijoms skirtų programavimo kalbų ir kitokio tipo įrankių sukurta nemažai (ColdFusion, Lasso, PHP, ASP ir t.t.), dauguma jų paremti tais pačiais principais kaip ir įprastinių aplikacijų įrankiai (struktūrinio, objektinio programavimo palaikymas, funkcijų, klasių bibliotekos ir t.t.). Pagrindinės ypatybės, skiriančios juos nuo kitų įrankių – HTML (ir kitų žymų kalbų) dokumentų formavimui pritaikyta sintaksė, bei funkcijų (klasių) bibliotekos, skirtos darbui WEB aplinkoje. Kitaip tariant, iki šiol kurtos WEB aplikacijų kūrimo priemonės apsiriboja programavimo užduočių automatizavimu.

Šis darbas išsiskiria ir yra naujas savo požiūriu į WEB aplikacijas, kadangi pagrindinis dėmesys skiriamas ne įvesties-išvesties priemonėms (HTML formavimui, HTTP užklausų valdymui ir t.t.), o pačių aplikacijų struktūrai ir veikimo pobūdžiui, kurį sąlygoja WEB architektūra. Kuriamoji kalba nepretenduoja tapti bendros paskirties programavimo kalba (kaip, pavyzdžiui, PHP), todėl gali pilnai išnaudoti WEB architektūros privalumus, optimizuoti programavimo procesą.

1.5. Praktinė nauda

Bet kuris programavimo įrankis, didinantis programuotojų darbo našumą, turi praktinės naudos: mažiau programuotojų per trumpesnį laiką gali sukurti didesnes ir sudėtingesnes aplikacijas. Atsižvelgiant į tai, kad WEB aplikacijų poreikis auga, ir nemažai įmonių užsiima WEB aplikacijų gamyba, tokio įrankio, kurį siekiama sukurti, nauda yra akivaizdi.

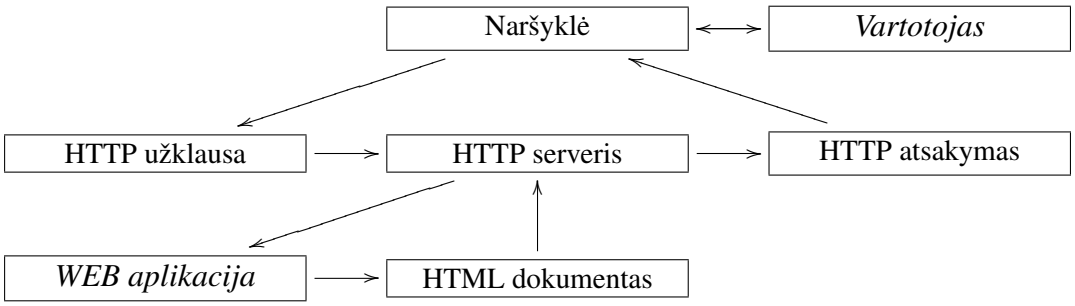
Šiame darbe realizuotas įrankis nebus tinkamas pilnaverčių aplikacijų kūrimui, todėl jo praktinis pritaikymas mažai tikėtinas. Kita vertus, atliktas tyrimas ir sukurta programavimo kalba (arba programavimo metodas) bus naudingi tiek vystant šį įrankį, tiek kuriant naujus.

Skyrius 2

WEB aplikacijų kūrimo technologijų analizė

2.1. WEB aplikacijų specifika

WEB aplikacijos veikia panašiu į kliento-serverio architektūros principu: vartotojai, naudodami klientines programas, pasiekia nutolusiame kompiuteryje veikiančią serverinę programą. Tačiau pačios WEB aplikacijos nėra nei klientinės, nei serverinės – serverio ir kliento vaidmenis atlieka HTTP serveris ir WWW naršyklė. Paveikslėlyje pateikiama WEB aplikacijų veikimo aplinka:



2.1: WEB aplikacijų vykdymo aplinka

Dėl tokio WEB aplikacijų vykdymo mechanizmo atsiranda esminiai WEB ir įprastų programų skirtumai:

- WEB aplikacijos paleidžiamos tik tada, kai vartotojas atlieka veiksmą, ir, pateikusios su tuo veiksmu susijusį rezultatą, baigia darbą.
- Informacija apie vartotojo atliktus veiksmus (pasirinktą nuorodą, laukeliuose įrašytas reikšmes ir t.t.) WEB aplikaciją pasiekia per aplinkos kintamuosius, kuriuos nustato HTTP serveris.
- Aplikacijos darbo rezultatų pateikimas, bei vartotojo sąsaja formuojama kaip HTML dokumentas, kurį aplikacija savo darbo pabaigoje įrašo į standartinį išvesties failą.

Tuo tarpu įprastinės aplikacijos vykdomos visos darbo sesijos metu; vartotojo sąsaja nėra griežtai suskaidyta į įvestį ir išvestį – aplikacija gali bet kuriuo metu pateikti ir gauti informaciją iš vartotojo. Pats informacijos gavimas bei pateikimas taip pat paprastesnis ir nereikalauja tarpinių duomenų formatų.

Šie skirtumai WEB aplikacijų kūrėjus skatina kurti ypatingus mechanizmus ir programavimo technikas, kurios leistų vartotojo atžvilgiu WEB aplikacijas priartintų prie įprastinių programų, bei palengvintų pačių aplikacijų programavimą.

2.1.1. Sesijos

Kadangi WEB aplikacijos paleidžiamos iš naujo kiekvieną kartą vartotojui atlikus veiksmą, tenka ieškoti būdų kaip išsaugoti programos būseną iki kito veiksmo; be to, aplikacija vienu metu gali naudotis keletas

vartotojų, todėl išsaugoti reikia su kiekvienu vartotoju susijusią programos būseną. Ši problema paprastai sprendžiama naudojant naršyklės priemones su svetaine susijusiai informacijai saugoti – slapukų mechanizmą (angl. cookie). Galimi du slapukų panaudojimo scenarijai:

- Programos būsena (t.y. kintamųjų reikšmės) kiekvieną kartą prieš programai baigiant darbą išsaugoma slapukuose ir, programai pradėjus vėl veikti, nuskaitoma. Šį mechanizmą paprasta realizuoti, ypač kai dauguma WEB programavimo įrankių suteikia priemones darbui su slapukais. Tačiau šis mechanizmas nėra sugus – vartotojai gali patys pakeisti naršyklėje įrašytas kintamųjų reikšmes, tokiu būdu priversdami programą elgtis ne taip kaip numatyta. Dėl šios priežasties dažniausiai renkamasis antras scenarijus.

- Su kiekvienu vartotoju susijusios programos būsenos saugomos serverio pusėje (duomenų bazėje ar failų sistemoje), o vartotojo naršyklėje išsaugomas identifikacinis raktas, pagal kurį aplikacija suranda ir atstato savo būseną. Toks mechanizmas pakankamai saugus (vartotojų raktai paprastai generuojami naudojant maišos funkcijas, surišami su IP adresais ir t.t., todėl mažai tikėtina, kad vartotojas sugebės apsimesti kitu tuo pat metu prisijungusiu vartotoju), tačiau sunkiau realizuojamas, kadangi tenka rūpintis papildomomis duomenų talpyklomis.

2.1.2. Klientinės dalies programavimas

Populiariausios naršyklės turi JavaScript interpretatorių, kuris gali vykdyti svetainėje patalpintas šia kalba parašytas programas. Ši naršyklių savybė leidžia WEB aplikacijas vartotojo atžvilgiu priartinti prie įprastinių, kuriant interaktyvius HTML puslapius. Tačiau pačių WEB aplikacijų veikimas nuo to beveik nesikeičia – vartotojo įvestis ir išvestis vykdoma tokiu pat principu, išlieka sesijų problema.

Kita vertus, dalį programos logikos galima perkelti į HTML puslapį, kuriame gali būti atliekamas įvestų duomenų patikrinimas, ar kitaip apdorojami įvedami ir pateikiami duomenys. Tačiau toks programos išskaidymas labai pavojingas, kadangi neįmanoma kontroliuoti vartotojo kompiuteryje veikiančios programos – t.y. sukuriamas potencialus saugumo spragų šaltinis.

Sparčiai plinta nauja JavaScript panaudojimo technika, vadinama AJAX (angl. Asynchronous JavaScript and XML). Šios technikos esmė yra tai, kad interaktyvus HTML puslapis, reaguodamas į vartotojo veiksmus (arba kitus įvykius), pats atlieka HTTP užklausas (t.y. aktyvuoja serveryje veikiančią WEB aplikaciją), perduodamas duomenis, susijusių su tuo įvykiu. Interpretuojant gautą užklausos rezultatą, atnaujinamos atitinkamos HTML dokumento dalys. Toks mechanizmas labai pagreitina aplikacijos darbą vartotojo atžvilgiu, tačiau programavimas tampa sudėtingesnis – tenka rūpintis daugeliu smulkių užklausų.

2.1.3. Darbas su HTML

Kiekviena WEB aplikacija kaip savo darbo rezultatą pateikia HTML dokumentą, todėl beveik visada naudojama viena iš keleto programavimo technikų, palengvinančių HTML (bei kitų formatų) dokumentų formavimą. Labiausiai paplitusios šios technikos:

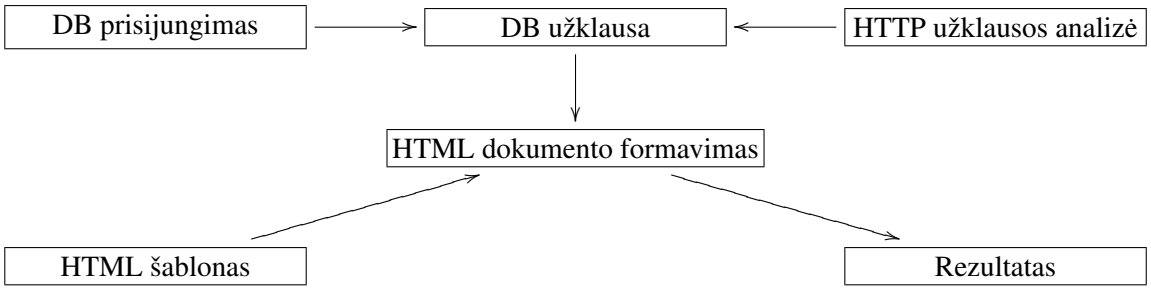
- Šablonai (angl. template). Aplikacijos pateikiami dokumentai paruošiami atskirai, dinamiškai besikeičiančias vietas išskiriant specialiomis žymomis. WEB aplikacija apdoroja šabloną, pažymėtose vietose įrašydama atitinkamas reikšmes, ir pateikia kaip rezultatą. Šis mechanizmas populiarus WWW svetainių programavime, kadangi svetainės išvaizda nepririšama prie programinio kodo ir gali būti lengvai keičiama, be to šablonus gali ruošti programavimo įgūdžių neturintys WEB dizaineriai.

- Programinio kodo įterpimas į dokumentą. Daugelio WEB aplikacijų programavimo kalbų sintaksė sukurta taip, kad programinis kodas tarsi įterpiamas į HTML dokumentą (PHP, ASP ir t.t.) – tai leidžia greitai ir paprastai formuoti HTML dokumentus, tačiau dokumentų išvaizda ir struktūra pririšama prie programinio kodo; be to, pats programinis kodas tampa sunkiai skaitomas.

- Trečioji technika sukurta kaip pirmų dviejų mišinys, siekiant padidinti programų našumą (šablonų apdorojimas reikalauja papildomų resursų), bei išspręsti programinio kodo organizavimo problemą. Naudojant šią techniką, programos kuriamos kaip klasių ar funkcijų bibliotekos; šablonai, kurie kuriami kalbomis, leidžiančiomis įterpti programinį kodą, tampa tų bibliotekų naudotojais.

2.2. Tipinės WEB aplikacijų užduotys

WEB aplikacijos išsiskiria ne tik savo kūrimo ypatybėmis, bet ir atliekamomis užduotimis. Dažniausiai šios užduotys apsiriboja informacijos surinkimu iš įvairių šaltinių (duomenų bazių, failų ir t.t.), minimaliu apdorojimu ir pateikimu vartotojui; arba iš vartotojo gautos informacijos apdorojimui ir talpinimui duomenų saugyklose. Pvz.:



2.2: Tipinės WEB aplikacijos veikimo schema

Toks užduočių paprastumas sąlygotas ne tik poreikių, bet ir galimybių: WEB aplikacijos paleidžiamos kiekvieną kartą vartotojui atlikus veiksmą, atlieka su tuo veiksmu susijusias užduotis ir pateikia rezultatą – t.y. reaguoja į veiksmą, todėl turi veikti greitai. Be to, aplikacija dažniausiai vienu metu naudojasi ne vienas vartotojas (vienu metu prisijungusių vartotojų skaičius gali siekti šimtus tūkstančių ir daugiau), todėl aplikacija turi optimaliai išnaudoti sistemos resursus.

Sudėtingesni uždaviniai (duomenų apdorojimas, analizė ir t.t.) paprastai perkeliami periodiškai paleidžiamoms programoms, kurios veikia ne būtinai WEB pagrindu.

2.3. Paplitę WEB aplikacijų kūrimo metodai ir priemonės

Nepaisant skirtumų, WEB aplikacijos dažniausiai kuriamos tais pačiais metodais kaip ir įprastinės. Atsižvelgiant į projekto dydį ir pobūdį, taip pat programuotojų įpročius, WEB aplikacijos gali būti kuriamos taikant bet kurią programavimo paradigmą. Architektūriniu požiūriu taip pat nėra apribojimų: nors WEB aplikacijos vykdomos serverio-kliento aplinkoje, pačios programos gali būti kitų aplikacijų klientai, veikti paskirstytai, naudoti vamzdžių ir filtrų mechanizmus ir t.t.

Įrankiams, kuriais kuriamos WEB aplikacijos, taip pat nėra apribojimų – praktiškai bet kuri programavimo priemonė leidžia pasiekti aplinkos kintamuosius, bei skaityti ir įrašyti tekstą į standartinį išvesties failą – t.y. atlikti minimalias operacijas, reikalingas aplikacijos veikimui. Tačiau, atsižvelgiant į WEB aplikacijų specifiką, bei dažniausiai pasitaikančias užduotis, sukurta nemažai specializuotų įrankių, palengvinančių programuotojo darbą.

2.3.1. WEB aplikacijų programavimo kalbos

Iki šiol, iš specializuotų WEB aplikacijų kūrimo priemonių dažniausiai buvo naudojamos WEB aplikacijų programavimo kalbos (PHP, Lasso ir t.t.). Dauguma šių kalbų beveik nesiskiria nuo įprastinių (ASP atveju kaip pagrindas naudojamos įprastinės kalbos) – t.y. turi įprastines kalbos konstrukcijas, klasių ar funkcijų bibliotekas, palaiko struktūrinį, objektinį programavimą ir t.t.

Pagrindinė tokių kalbų ypatybė – sintaksė, leidžianti programinį kodą įterpti formuojamo HTML dokumento viduje. Tokia sintaksė supaprastina ir pagreitina HTML dokumento – WEB aplikacijos vartotojo sąsajos – formavimo programavimą. Kita WEB programavimo kalbų ypatybė – klasių ar funkcijų bibliotekos, skirtos darbui su dažniausiai WEB aplikacijose naudojamomis technologijomis: http užklausų valdymu, duomenų bazėmis, operacijomis su tekstu ir panašiai.

2.3.2. WEB aplikacijų karkasai (angl. WEB framework)

WEB aplikacijų karkasai – palyginus, nauji, tačiau sparčiai populiarėjantys įrankiai. Šių įrankių esmė – automatizuotas tipinių WEB aplikacijų užduočių programavimas. Tipinės užduotys – tai HTML dokumentų formavimas, duomenų talpinimas ir nuskaitymas iš duomenų bazių, darbas su HTTP užklausomis, HTTP formų apdorojimas ir t.t. Kadangi HTML dokumento formavimas perkeliamas karkaso funkcijoms, WEB programavimo kalbų sintaksinės ypatybės netenka prasmės, todėl karkasai dažniausiai kuriami bendros paskirties programavimo kalbomis (Python, Ruby).

WEB karkasų sukurta pakankamai daug ir skirtingų, tačiau galima išskirti daugelyje pasitaikančias bendras ypatybes:

- Duomenų struktūros ir veiksmai su jomis aprašomi atskirai nuo kitų programos dalių. Karkasas pagal šiuos aprašus, vadinamus modeliu, generuoja duomenų bazės užklausas tiek lentelių kūrimui, tiek duomenų paėmimui ir talpinimui. Programuotojas su duomenų įrašais dirba kaip su programiniais objektais – atliekamas taip vadinamas objektinis-reliacinis transliavimas.
- Puslapių ar puslapių dalių generavimas aprašomas atskirai nuo kitų programos dalių, nurodant šablonus, pagal kuriuos generuojamas dokumentas, bei papildomus veiksmus. Tokios atskirtos dalys vadinamos vaizdais.
- Daugelis karkasų taip pat turi funkcijas automatiniam vartotojo sąsajos generavimui – t.y. pagal aprašytus duomenų modelius automatiškai paruošiamos ir apdorojamos HTML formos, leidžiančios talpinti ir redaguoti duomenis duomenų bazėje.

Aprašytas duomenų struktūros ir atvaizdavimo atskyrimas yra MVC (angl. Model-view-controller) [2] architektūrinio šablono pagrindas, kuriuo remiasi dauguma karkasų.

2.4. WEB aplikacijų kūrimo problemos

Pagrindinių WEB aplikacijų kūrimo problemų – darbo sesijų ir vartotojo sąsajos ribotumo – priežastis yra tai, kad WEB technologija buvo kuriama ne kaip interaktyvių aplikacijų platforma, o kaip informacijos apsikeitimo įrankis. Šių problemų sprendimas tapo neatsiejama kiekvienos WEB aplikacijos dalimi, iš dalies įtakojančia ir pačios aplikacijos logiką. Specializuoti įrankiai palengvina šių problemų sprendimą, tačiau neretai sudaro papildomų sunkumų.

2.4.1. WEB aplikacijų programavimo kalbų problemos

Pagrindinis WEB aplikacijų programavimo kalbų keliamas sunkumas atsiranda iš programinio kodo ir HTML dokumento samplaikos. Nepatyrę programuotojai dažnai nesistengia atskirti programinio ir HTML kodo, dėl ko programos tampa sunkiai skaitomos, neaiškios struktūros. Siekiant atskirti programinį kodą, naudojamos įvairios technikos (dažniausiai šablonų mechanizmai) – tai reikalauja ne tik papildomų programuotojo pastangų, bet ir papildomų kompiuterio resursų.

Kita dažnai pasitaikanti problema – programinio kodo kartojimas. WEB aplikacijų kalbų bibliotekos suteikia priemones darbui su technologijomis, bet ne uždavinių sprendimui. Programuotojams tenka kiekvieną kartą iš naujo programuoti formų kūrimą ir apdorojimą, sesijos kintamuosius ir t.t. – t.y. spręsti praktiškai kiekvienoje aplikacijoje pasikartojančias užduotis.

2.4.2. Aplikacijų karkasų problemos

WEB aplikacijų karkasai išsprendžia daugumą WEB programavimo kalbų problemų, automatizuodami dažniausiai pasitaikančių uždavinių sprendimą. Kita vertus, karkasai paprastai automatizuoja visos aplikacijos veikimo programavimą, programuotojui palikdami tik duomenų modelio ir vaizdo aprašymą. Dėl šios priežasties nestandartinių aplikacijų programavimas tampa pakankamai sudėtingas ar neįmanomas.

Skyrius 3

WEB aplikacijų programavimo kalbos specifikacijos

3.1. Programavimo kalbai keliami reikalavimai

Atsižvelgiant į nagrinėtų WEB aplikacijų kūrimo priemonių teikiamas galimybes ir trūkumus, galima suformuoti reikalavimus, keliamus kuriamajai kalbai:

- Šia kalba užrašytos programos turi būti lengvai skaitomos, prižiūrimos ir plečiamos, kadangi daugumą WEB aplikacijų tenka dažnai keisti ir pildyti, taikantis prie besikeičiančių sąlygų. Šį reikalavimą galima patenkinti ir įprastinėmis priemonėmis, tačiau tam reikalingos papildomos programuotojo pastangos bei žinios.
- Tipinės WEB aplikacijų funkcijos turi tapti kalbos dalimi. Taip programuotojas, mokėdamas šią kalbą, galės koncentruotis į pačios aplikacijos logiką, o ne į nuolat besikartojančių uždavinių sprendimą. Kita vertus, kalba neturi primesti savo sprendimų – t.y. turi palikti kiek įmanoma daugiau laisvės programuotojo poreikių įgyvendinimui.
- Kadangi pagrindinė WEB aplikacijų terpė yra Internetas, ypatingas dėmesys turi būti skirtas saugumui. Aplikacijos saugumas nėra vien realizacijos problema – į saugumą turi būti atsižvelgiama visuose aplikacijos kūrimo ir diegimo etapuose. Programavimo kalba savo ruožtu gali suteikti priemones bei sprendimų šablonus saugių aplikacijų kūrimui.
- WEB aplikacijos turi optimaliai išnaudoti kompiuterio resursus, kadangi vienoje sistemoje veikiančia aplikacija vienu metu gali naudotis labai didelis skaičius vartotojų. Nors programos našumas daugiausia priklauso nuo projekto ir realizacijos, programavimo kalba savo pobūdžiu gali skatinti našesnių programų kūrimą. Taip pat, atsižvelgiant į procesorių vystymosi tendencijas (keleto branduolių procesoriai), kalba turėtų leisti nesudėtingai kurti daugelio srautų programas.

3.2. Bendras sukurto aplikacijų kūrimo metodo aprašymas

Vizuali programavimo kalba (angl. VPL – Visual Programming Language) – tai programavimo kalba, kuria programos išreiškiamos grafiškai – t.y. dėstant elementus dvimatėje ar trimatėje erdvėje. Tokio tipo programavimo kalbos kuriamos nuo 1965 metų [7], tačiau didesnio dėmesio už mokslo įstaigų ribų iki šiol nesulaukė. Galima tokio nepopuliarumo priežastis yra tai, kad sudėtingi algoritmai ir matematinės formulės lengviau ir suprantamiau išreiškiami tekstiniu pavidalu.

Kita vertus, duomenų srauto architektūros programavime grafinės kalbos dominuoja, kadangi grafinis duomenų apdorojimo ir perdavimo tinklų vaizdavimas yra aiškesnis nei tekstinis. Pats duomenų srauto programavimo principas taip pat nėra paplitęs už mokslo ar specifines programas kuriančių įstaigų ribų.

Atsižvelgiant į tipines WEB aplikacijų užduotis bei specifiką (duomenų išgavimas – apdorojimas – pateikimas), daroma prielaida, kad WEB aplikacijų kūrimas galėtų būti efektyvesnis taikant duomenų srautų architektūrinius principus. Remiantis šia prielaida, WEB aplikacijų programavimo kalba kuriama kaip duomenų srautų kalba. Siekiant užtikrinti paprastą ir patogų programavimą, kuriama vizuali kalba, paliekant galimybę matematines išraiškas, bei kitas sudėtingesnes struktūras užrašyti tekstiniu pavidalu.

3.2.1. Kalbos dalys ir taisyklės

Kalboje naudojami trijų rūšių elementai:

- Instrukcijos – nurodo kokie veiksmai atliekami su duomenimis (nuskaitymas, įrašymas, apdorojimas, sąlygų tikrinimas ir t.t.);
- Ryšiai – nurodo kaip duomenys perduodami tarp instrukcijų.
- Konstantos – programoje įrašomi pradiniai duomenys, pateikiami kaip instrukcijų parametrai.

Kintamųjų aprašų kalboje nėra: duomenys perduodami tiesiogiai iš instrukcijos į instrukciją – tai duomenų srauto architektūros ypatybė.

Pagrindinės programų užrašymo ir veikimo taisyklės:

- Kiekviena instrukcija turi nulį arba daugiau įvardintų duomenų įėjimų (parametrų), kurie gali būti privalomi arba neprivalomi.
- Kiekviena instrukcija turi vieną duomenų išėjimą (rezultatą), kuris gali būti naudojamas kaip kitos (arba tos pačios) instrukcijos parametras. Srauto valdymo instrukcijos gali turėti daugiau nei vieną rezultatą, tačiau visi vienos instrukcijos rezultatai turi būti vienodi.
- Konstantos prijungiamos ir perduodamos kaip instrukcijų parametrai.
- Instrukcija aktyvuojama ir grąžina rezultatą tik tada, kai kita instrukcija bando pasiekti savo parametrą, kuris yra šios instrukcijos rezultatas.
- Viena instrukcija pažymima kaip pradinė ir aktyvuojama aplikacijos paleidimo metu.

3.2.2. Instrukcijų rinkiniai

Kalbos taisyklėse nurodyta, kad visi veiksmai, įskaitant srauto valdymą, realizuojami instrukcijų pagalba. Toks sprendimas leidžia programavimo kalbą pritaikyti praktiškai bet kuriai sričiai, kuriant skirtingus instrukcijų rinkinius. WEB aplikacijoms pritaikytoje kalbos versijoje turi būti realizuotos tiek bendros paskirties instrukcijos, tiek skirtos tipinėmis WEB aplikacijų užduotimis:

- Srauto valdymas – instrukcijos, skirtos sąlyginiam duomenų kanalo parinkimui, apjungimui, ciklams ir t.t.
- Įvestis-išvestis – darbo su failais instrukcijos.
- Išraiškų sprendimas – instrukcijos, skaičiuojančios tekstiniu pavidalu užrašytas formules.
- Darbas su duomenų bazėmis – instrukcijos, skirtos prisijungimui prie įvairių duomenų bazių valdymo sistemų, ir leidžiančios atlikti užklausas.
- Darbas su HTML dokumentais – HTML ir kitokių dokumentų šablonų mechanizmo instrukcijos.
- Darbas su CGI – instrukcijos, skirtos pasiekti ir valdyti HTTP užklausos informaciją (URL adresas, formų duomenys, slapukai ir t.t.).

Kuriant vis daugiau aplikacijų, paaiškės kokie instrukcijų rinkiniai dar reikalingi, kurie iš egzistuojančių nereikalingi – t.y. ši kalbos dalis bus pakankamai dinamiška.

3.3. Kalbai keliamų reikalavimų tenkinimas

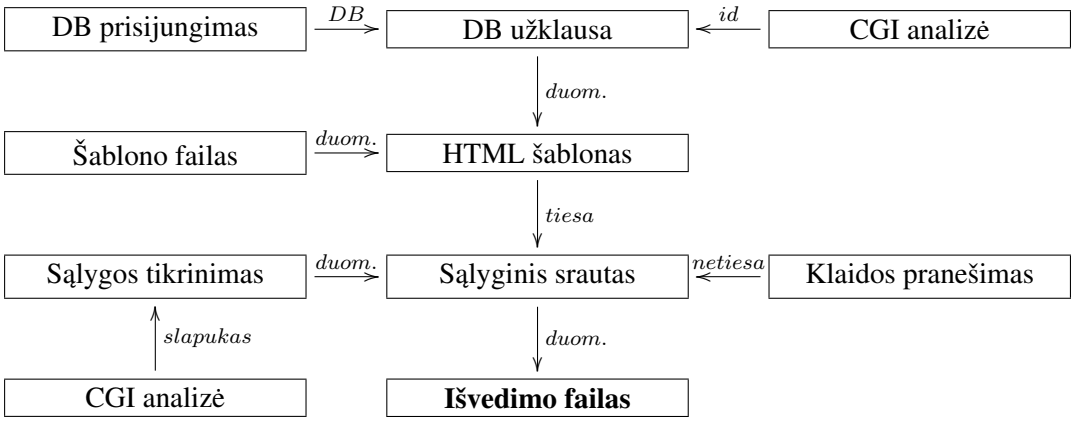
- Skaitomumas, prižiūrimumas. Pagrindinė prastesnio įprastinėmis WEB programavimo kalbomis parašytų programų skaitomumo priežastis – programinio ir HTML kodo apjungimas. Šioje kalboje tai neįmanoma, kadangi programinio kodo apskritai nelieka. Tačiau atsiranda kitas pavojus – Doičo riba [8], kuri teigia, kad grafiškai kuriamoje programoje negali būti daugiau nei 50 elementų. Kita vertus, mažai tikėtina, kad bus kuriamos tokio dydžio WEB aplikacijos.
- Funkcionalumas. Kuriamosios kalbos funkcionalumas tiesiogiai priklauso nuo instrukcijų rinkinio. Labai sunku iš karto sudaryti optimalų instrukcijų rinkinį, todėl ši kalbos dalis bus vystoma ilgą laiką: palaipsniui bus atsisakoma nenaudojamų instrukcijų, ir įvedamos naujos, kurių prireiktų kuriant įvairias aplikacijas.
- Saugumas. Dažniausiai pasitaikančių WEB aplikacijų saugumo spragų – SQL injekcijų ir tarpsaitinių skriptų (angl. XSS – Cross-site scripting) [9] – galima išvengti tinkamai tikrinant duomenis, su kuriais dirba aplikacija. Nors pati programavimo kalba to negali padaryti, tinkamos instrukcijos, bei aiškus vizualus duomenų srautų atskyrimas gali padėti programuotojui rašyti saugesnes aplikacijas.
- Našumas. Šia kalba rašomų programų našumas daugiausia priklauso nuo instrukcijų realizacijos, kadangi programuotojas nurodo tik kokios instrukcijos bus atliktos, pats nerašydamas programinio kodo. Taisyklėse nurodoma, kad vykdomos tik tos instrukcijos, kurių rezultatai reikalingi, taip išvengiant bereikalingų veiksmų. Realizuojant šios kalbos transliatorių reikia atsižvelgti į tai, kad kalba yra duomenų srautų, kas leidžia ja parašytas programas automatiškai skaidyti į atskirus srautus.

3.4. Teorinis metodo testavimas

Vizualių duomenų srautų programavimo kalbų sukurta pakankamai nemažai, tačiau dauguma jų orientuotos į siauras pritaikymo sritis. Pagrindinės tokių kalbų taikymo sritys – matematiniai skaičiavimai, darbas su grafika, robotų valdymas. Bene labiausiai į kuriamąją panaši egzistuojanti kalba – cantata [10] – skirta grafinių vaizdų apdorojimui, tuo tarpu WEB aplikacijoms skirtų, arba pritaikomų kalbų rasti nepavyko. Todėl kyla klausimas, ar tokio tipo kalba apskritai įmanoma kurti WEB aplikacijas.

Norint atsakyti į šį klausimą, užrašomos dvi aplikacijos, atliekančios duomenų paėmimą iš duomenų bazės, pateikimą vartotojui, minimalų apdorojimą ir išsaugojimą duomenų bazėje. Taip pat bandoma imituoti interpretatoriaus veikimą vykdant šias aplikacijas.

3.4.1. Duomenų pateikimo aplikacija



3.1: Duomenų pateikimo aplikacija

1. Aktyvuojama išskirtoji instrukcija – „išvedimo failas“, – kuri įrašo duomenis į standartinį išvesties failą. Duomenys gaunami iš parametro, todėl aktyvuojama prijungta sąlyginio srauto instrukcija.

2. Sąlyginio srauto instrukcija grąžina duomenis, gautus iš „tiesa“ arba „netiesa“ parametro, priklausomai nuo duomenų parametro, todėl aktyvuojama sąlygos tikrinimo instrukcija.

3. Sąlygos tikrinimo instrukcijoje išraiškos pavidalu įrašytas predikatas, kurio parametras gaunamas iš instrukcijos parametro, todėl kviečiama CGI analizės instrukcija.

4. CGI analizės instrukcijoje įrašytas slapuko pavadinimas, kurio reikšmę instrukcija nuskaito ir grąžina kaip savo veikimo rezultatą.

5. Jei sąlyga buvo nepatenkinta, „sąlyginis srautas“ aktyvuoja klaidos pranešimo instrukciją, kuri grąžina klaidos tekstą. Instrukcija „išvedimo failas“ gauna šį tekstą, įrašo į standartinę išvestį; programa baigia darbą.

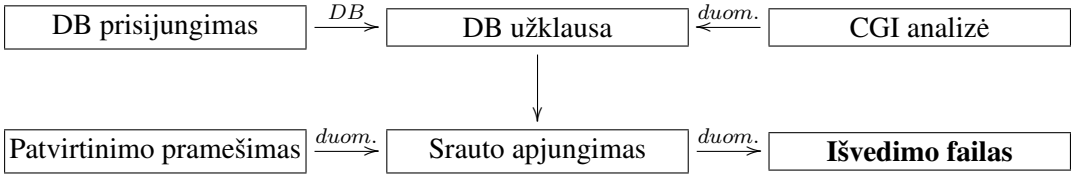
6. Jei sąlyga buvo patenkinta, aktyvuojama HTML šablono instrukcija, kuri pagal pateiktą šabloną formuoja HTML dokumentą. Instrukcija bando apdoroti šabloną – aktyvuojama šablono failo instrukcija.

7. Šablono failo instrukcija iš nurodytos vietos nuskaito HTML šabloną ir grąžina kaip savo rezultatą.

8. Šabloną apdorojanti instrukcija aptinka vietą, kurioje turi būti įrašyti duomenys – aktyvuojama DB instrukcija, kurioje įrašyta SQL užklausa. Šiai instrukcijai reikalingas duomenų bazės prisijungimas ir įrašo identifikatorius: aktyvuojamos DB prisijungimo ir CGI analizės instrukcijos.

9. Prisijungiama prie duomenų bazės, gaunamas reikalingas identifikatorius, SQL užklausa įvykdoma. Šablono instrukcija, gavusi reikalingus duomenis, baigia formuoti HTML dokumentą, kuris pasiekia išvedimo instrukciją. Programa baigia darbą.

3.4.2. Duomenų talpinimo aplikacija



3.2: Duomenų talpinimo aplikacija

1. Aktyvuojama išskirtoji instrukcija – „išvedimo failas“. Aktyvuojama su šios instrukcijos parametru sujungta srauto apjungimo funkcija.

2. Srauto apjungimo funkcija iš eilės kreipiasi į visus prijungtus parametrus ir grąžina paskutiniojo reikšmę. Aktyvuojama DB užklausos instrukcija.

3. DB užklausos instrukcijoje įrašyta SQL užklausa, talpinanti duomenis nurodytoje lentelėje. Duomenys gaunami kaip parametras – aktyvuojamos CGI analizės ir DB prisijungimo instrukcijos.

4. DB prisijungimo instrukcija jungiasi prie duomenų bazės ir grąžina jungties nuorodą.

5. CGI analizės instrukcija grąžina nurodyto HTML formos lauko reikšmę – suveikia SQL užklausa. Papildomo formos duomenų tikrinimo nereikia, kadangi instrukcija realizuota taip, kad išoriniai duomenys nerašomi į SQL užklausos kodą. Tai reiškia, kad šia kalba parašytos programos bus savaime apsaugotos nuo SQL injekcijų atakų.

6. Srauto apjungimo instrukcija aktyvuoja kitą parametrą – suveikia instrukcija „patvirtinimo pranešimas“, kuri nuskaito ir grąžina pranešimą vartotojui. Duomenys pasiekia išvedimo failo instrukciją; programa baigia darbą.

3.4.3. Teorinio testavimo rezultatai

Tai, kad pavyko užrašyti ir „įvykdyti“ sugalvotas programas, dar neįrodo, kad ši programavimo kalba bus tinkama bet kokioms WEB aplikacijoms. Kalbos tinkamumą įmanoma įvertinti tik kuriant realias aplikacijas, o tam reikia realizuoti bent minimalias priemones, leidžiančias tai daryti.

Testavimas buvo naudingas ne tik tuo, kad leido paneigti teiginį, kad tokio tipo kalba iš principo nėra tinkama WEB aplikacijoms. Testavimas parodė kaip galėtų veikti (tuo pačių ir būti realizuotas) šios kalbos interpretatorius (ar kompiliatorius). Taip pat buvo išskirta keletas būtiniausių instrukcijų, kurios bus įtrauktos į pradinį kalbos variantą.

Skyrius 4

WEB aplikacijų programavimo kalbos realizacija

4.1. Programavimo kalbos realizacijos projektas

4.1.1. Programavimo priemonės specifikacijos

Programų vykdymas

Norint vykdyti kompiuteriu kuriamąja kalba parašytas programas, turi būti sukurta programa (ar programų rinkinys), vadinama transliatoriumi, kuri šia kalba užrašytas programas verstų į vykdymo kompiuteriui tinkamą formą. Galimi du tokios sistemos modeliai: kompiliatorius ir interpretatorius:

- Kompiliatorius šia kalba užrašytas programas transformuotų į procesoriaus instrukcijų rinkinį (mašininį kodą) ir paruoštų paleidžiamąjį failą, kuris būtų vykdomas tiesiogiai.
- Interpretatorius šia kalba užrašytas programas vykdytų iš karto, analizuodamas jų kodą ir atlikdamas atitinkamus veiksmus.

Pagrindiniai skirtumai tarp šių metodų – vykdymo greitis ir programos pernešamumas. Sukompiliuotos programos veikia daug kartų greičiau, kadangi nevykdoma papildoma analizė, tačiau jų pernešimas į kitas platformas reikalauja papildomų pastangų. Interpretatoriaus atveju programos be papildomų pakeitimų gali būti paleidžiamos kiekvienoje platformoje, kurioje veikia interpretatorius. WEB aplikacijos paprastai kuriamos ir leidžiamos skirtingose platformose, todėl interpretatoriaus modelio pasirinkimas yra labiau priimtinas. Našumo atžvilgiu neturėtų būti didelių nuostolių, kadangi pati kalba nėra sudėtinga ir interpretatorius turėtų veikti pakankamai greitai.

Pačios WEB aplikacijos taip pat talpinamos skirtingų platformų serveriuose, naudojant skirtingus HTTP serverius, todėl kuriamosios kalbos interpretatorius turi veikti daugelyje platformų ir būti suderinamas su bet kuriuo HTTP serveriu (Apache, ISS, lighttpd ir t.t.).

Paprasčiausias būdas sukurti programavimo kalbos interpretatorių, kurio pagalba HTTP serveris galėtų leisti WEB aplikacijas, yra parašyti programą, kuri per standartinę įvestį nuskaitytų interpretuojamos programos tekstą, ir į standartinę išvestį įrašytų HTTP antraštes bei HTML dokumentą, kurį suformavo WEB aplikacija. Tuomet HTTP serverio konfigūracijoje nurodoma, kad tam tikro tipo failus (šiuo atveju kuriamosios kalbos programas) reikia vykdyti naudojant nurodytą interpretatorių (naudojama CGI technologija). Egzistuoja ir kitų, našesnių, WEB aplikacijų leidimo būdų (FastCGI, HTTP serverio moduliai), tačiau pradiniame etape į juos atsižvelgti nėra poreikio.

Programų kūrimas

Programų užrašymui taip pat reikalingas specialus įrankis: kalba nėra tekstinė, todėl įprasti teksto redaktoriai netinka. Tai turėtų būti grafinis diagramų redaktorius, leidžiantis manipuliuoti objektais, simbolizuojančiais

instrukcijas, užrašyti konstantas, bei nurodyti ryšius tarp jų. Taip pat šis redaktorius turėtų turėti integruotą interpretatorių, kuris leistų testuoti kuriamas programas.

Daugiaplatformiškumas nėra būtinas, tačiau, norint, kad kalba paplistų tarp įvairiose platformose dirbančių programuotojų, tokia savybė būtų naudinga.

Siekiant sukurti patogų įrankį, galimos įvairios papildomos funkcijos: duomenų bazių projektavimas ir kūrimas, automatinis programų talpinimas serveryje, HTML šablonų ruošimas ir t.t. – t.y. galėtų būti sukurta pilnavertė WEB aplikacijų kūrimo aplinka.

4.1.2. Programavimo priemonės projektas

Aplikacijų užrašymo formatas

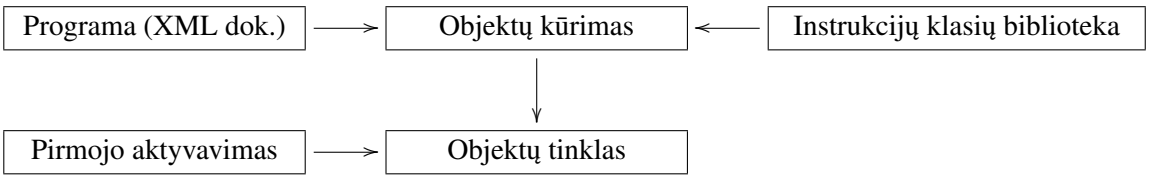
Kuriant aplikacijas, generuojama dviejų rūšių informacija: programos logika (instrukcijos, parametrai, duomenų ryšiai) ir išvaizda (grafinis elementų išdėstymas, programos komentarai). Interpretatoriui veikti reikalinga tik logika, todėl būtų optimalu naudoti du formatus: skirtą programuotojui ir skirtą interpretatoriui. Kita vertus, du formatai įneštų papildomo sudėtingumo, o dydžio ir greičio išlošimas pradiniame etape nėra svarbus, todėl programos logika ir vaizdas bus saugomi viename faile.

Programavimo kalba yra pradinėje gyvavimo stadijoje, todėl tikėtina, kad aplikacijų užrašymui naudojamos duomenų struktūros ar jų dalys dažnai keisis – formatas turi leisti atskiras dalis keisti neįtakojant kitų. Be to, saugomi įvairių tipų duomenys (konstantos gali būti ne tik tekstinės), todėl formatas turi būti pakankamai lankstus. XML tenkina šiuos reikalavimus, nesukeldamas papildomų sunkumų programuotojui, tačiau XML apdorojimas reikalauja papildomų kompiuterio resursų. Optimizuojant sistemą, galėtų būti sukurtas binarinis formatas, tačiau pradiniuose etapuose tinkamesnis XML pasirinkimas.

Pasiekus kalbos vystymo stadiją, kai formatas nebesikeičia, bus galima formaliai aprašyti XML schemą [11]. Iki tol formatas aprašomas neformaliai – pavyzdžiais ir komentarais.

Aplikacijų vykdymo įrankis

Programavimo kalbos paprastumas leidžia sukurti paprastą, lengvai plečiamą ir našų interpretatorių: kiekviena instrukcija ir konstanta realizuojama programiniu objektu, ryšiais sujungtu su kitais objektais. Programos užkrovimo metu sukuriamas objektų tinklas. Paleidus programą, aktyvuojamas pirmasis objektas, kuris, jei reikia aktyvuoja kitus – taip susidaro objektų aktyvavimo lavina.



4.1: Interpretatoriaus veikimo schema

Interpretatoriaus programą realizuojant pagal pateiktą schemą, atsiranda dvi nepriklausomos dalys: statinė ir dinaminė. Statinė dalis atsakinga už objektų tinklo sukūrimą pagal pateiktą programą ir pirmojo objekto aktyvavimą. Dinaminėje dalyje realizuojama instrukcijų klasių biblioteka, kuri gali keistis priklausomai nuo kalbos modifikacijos.

Siekiant išgauti kuo didesnį našumą, objektų tinklo kūrimas ir paleidimas galėtų būti atskirti – t.y. iš anksto paruošiami objektų tinklai ir aktyvuojami tada, kai reikia paleisti aplikaciją (kuriamas savotiškas aplikacijų serveris). Kita optimizacijos galimybė – instrukcijas realizuoti taip, kad nepriklausomus parametrus patiekiantys objektai būtų aktyvuoti atskiruose srautuose. Tikėtina, kad šios optimizacijos leistų programoms veikimo greičiu ne tik nenusileisti vieno srauto kompiliuojamoms programoms, bet, kai kuriais atvejais, jas lenkti.

Aplikacijų kūrimo įrankis

Siekiant gauti veikiantį rezultatą per maksimaliai trumpą laiką (kuriant kalbą buvo reikalingos priemonės rašyti bandomąsias programas), šis įrankis buvo realizuotas trečiosios šalies programos pagalba. Buvo pasirinkta Dia [12] programa, skirta diagramų kūrimui, bei sukurtas konverteris, verčiantis šia programa sukurtas diagramas į programų vykdymo aplinkos formatą.

4.1.3. Programavimo priemonės realizacijos projektas

Įrankių pasirinkimas

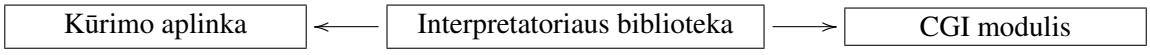
Programavimo kalbos pasirinkimą lėmė tai, kad kuriamas kitos kalbos interpretatorius – t.y. jei pats interpretatorius bus parašytas interpretuojama kalba, kuriamąja kalba parašytos programos veiks labai lėtai – t.y. interpretatorius turi būti realizuotas kompiliuojama kalba. Taip pat pasirinkta kalba turi palaikyti objektinį programavimą, bei veikti daugelyje platformų. Kaip geriausiai atitinkanti šiuos reikalavimus buvo pasirinkta C++ programavimo kalba.

Kadangi C++ kalboje nėra priemonių darbui su XML, duomenų bazėmis ir t.t., reikalingos papildomos bibliotekos. Siekiant supaprastinti programavimą ir diegimą, tai galėtų būti viena biblioteka. Viena geriausiai tenkinančių šiuos reikalavimus C++ bibliotekų, taip pat veikianti daugelyje platformų, yra Qt [13]. Šioje bibliotekoje yra priemonės darbui su UTF simbolių eilutėmis, XML, populiariausiomis duomenų bazių valdymo sistemomis, tinklu – t.y. dažniausiai WEB aplikacijose naudojamomis technologijomis. Taip pat yra priemonės darbui su grafine vartotojo sąsaja, todėl programų kūrimo įrankio realizacijai taip pat bus pasirinktas šis kalbos-bibliotekos derinys.

Qt bibliotekos pasirinkimo neigiama pusė – licencijavimas. Nemokama Qt versija platinama pagal GPL licenciją, kuri verčia visas biblioteką naudojančias programas taip pat būti platinamas pagal šią licenciją. Komercinė Qt versija, leidžianti laisvai pasirinkti savo programos licenciją, yra pernelyg brangi, todėl aplikacijų kūrimo ir vykdymo programos turės būti platinamos pagal GPL licenciją. Kyla klausimas, ar šia sistema sukurtos WEB aplikacijos galės būti platinamos pagal kitokią licenciją – skirtingai interpretuojant GPL, galimi skirtingi atsakymai. Kalbos išpopuliarėjimo atveju, siekiant išvengti nesusipratimų, interpretatorius turės būti perrašytas nenaudojant Qt bibliotekos.

Programiniai moduliai

Interpretatorius bus naudojamas ne tik serveriuose, WEB aplikacijų leidimui, bet ir jas kuriančių programuotojų kompiuteriuose, kaip aplikacijų kūrimo aplinkos dalis, skirta aplikacijų testavimui ir klaidų paieškai. Dėl šios priežasties racionalu būtų interpretatorių kurti kaip nepriklausomą biblioteką, prijungiamą tiek prie serverinio CGI modulio ar aplikacijų serverio, tiek prie programų kūrimo aplinkos. Pradiniame etape galima išskirti tokius programinius modulius:



4.2: Programiniai moduliai

4.2. Programavimo kalbos ir priemonės būsena

4.2.1. Dabartinė būsena

Sukurta programavimo kalbos specifikacija pilnai nusako kalbos struktūra ir taisykles, taip pat ir pačių programų veikimo principą – t.y. pagrindinė programavimo kalbos kūrimo dalis baigta. Instrukcijų rinkinių aprašymas nebaigtas, kadangi patys instrukcijų rinkiniai nėra nusistovėję (žr. programuotojo dokumentaciją – instrukcijų aprašymai pateikiami kartu su jas realizuojančių klasių aprašymais).

Programavimo priemonė – t.y. aplikacijų kūrimo ir vykdymo aplinkos – taip pat pradinėje stadijoje. Aplikacijų kūrimo aplinkos vaidmenį atlieka trečiosios šalies diagramų kūrimo programa kartu su pateikiamu konverteriu, verčiančiu šios programos formato failus į vykdymo aplinkos failus.

Vykdymo aplinka (arba interpretatorius) – sudaryta iš statinės ir dinaminės dalies. Statinė – dalis, atsakinga už aplikacijos objektų tinklo paruošimą ir aktyvavimą – suprojektuota ir realizuota. Dinaminės dalies realizacija nuolat pildoma. Šiuo metu realizuotų funkcijų pakanka darbui su duomenų apsikeitimu naudojant HTTP protokolą, duomenų bazėmis (visoms DBVS, kurias palaiko Qt biblioteka), HTML šablonais, reguliariosiomis išraiškomis.

Praktiniu kalbos pritaikymo pavyzdžiu gali pasitarnauti pirmoji šia kalba kuriama WEB svetainė [14]. Nors svetainė dar nebaigta (trūksta dalies turinio administravimo posistemės), galima pastebėti, kad kai kurių užduočių realizacija yra žymiai paprastesnė lyginant su įprastinėmis kalbomis (pvz. formos duomenų įrašymas į duomenų bazę ir atvirkščiai); kai kurių – sunkesnė, ir reikalauja ieškoti apėjimo būdų. Prieduose pateikiamos kuriamos svetainės programos (Dia formatu).

4.2.2. Tolimesni žingsniai

Artimiausi programavimo kalbos ir įrankių kūrimo žingsniai:

- Toliau kuriamos ir tobulinamos instrukcijos, leidžiančios kurti sudėtingesnes WEB aplikacijas.
- Sukurta grafinė aplikacijų kūrimo aplinka, leidžianti kurti ir testuoti WEB aplikacijas.
- Pasirinkta viena ar kelios egzistuojančios WEB aplikacijų kūrimo priemonės ir jomis sukurta tipinė WEB aplikacija; analogiška aplikacija sukurta kuriamąja kalba. Tai leistų palyginti kuriamąją kalbą su egzistuojančiomis, išskirti trūkumus ir privalumus, bei nustatyti tolimesnio vystymo kryptis.

Tolimesni žingsniai priklausys nuo poreikių: galbūt teks perrašyti aplikacijų vykdymo aplinką nenaudojant Qt bibliotekos, sukurti našų aplikacijų serverį, tobulinti aplikacijų kūrimo aplinką ir t.t. Kadangi projektas turės būti platinamas pagal GPL licenciją, tikėtina parama iš kitų programuotojų, savanoriškai prisidėsiančių prie projekto.

Skyrius 5

Išvados

Pirminis darbo tikslas – WEB aplikacijų programavimo kalba – pasiektas: sukurta ir aprašyta programavimo kalba, orientuota į WEB aplikacijų kūrimą, naudojant duomenų srautų principus. Kalbos veiksmingumas teoriniame lygmenyje pagrįstas: aprašytos dvi WEB aplikacijos ir sumodeliuotas jas vykdančio interpretatoriaus veikimas. Kalbos atitikimas iškeltiems reikalavimams taip pat grindžiamas teoriniame lygmenyje: vadovaujamasi teiginiais, kad duomenų srautais paremtas programavimas yra tinkamas WEB aplikacijų kūrimui dėl pačių aplikacijų specifikos; vizuali kalba yra lengviau skaitoma ir suvokiama, todėl programos bus lengviau prižiūrimos ir plečiamos.

Praktinis kalbos tinkamumo patikrinimas susijęs su antriniu darbo tikslu – programine kalbos realizacija. Šis patikrinimas atliktas dalinai: kuriama minimali WEB aplikacija, panaudojant realizuotą, tačiau nuolat tobulinamą instrukcijų rinkinį. Savaime suprantama, tokio patikrinimo rezultatai nėra konkretūs, tačiau leidžia įsitikinti, kad kuriamąja kalba įmanoma kurti realias, veikiančias WEB aplikacijas, be to, kai kurios WEB aplikacijų užduotys kuriamąja kalba realizuojamos daug paprasčiau nei tradicinėmis WEB aplikacijų kūrimo priemonėmis.

Įgyvendinus artimiausius uždavinius – t.y. ištobulinus instrukcijų rinkinius, bei sukūrus WEB aplikacijų kūrimo aplinką, bus galima kurti sudėtingas WEB aplikacijas, kurios leis sukurtąją kalbą realiais pavyzdžiais palyginti su kitomis WEB aplikacijų kūrimo priemonėmis. Jei paaiškėtų, kad sukurtoji kalba yra pranašesnė už kitas priemones, būtų galima toliau vystyti kalbą, bei tobulinti aplikacijų kūrimo aplinką. Kitu atveju gali tekti kalbą kurti iš naujo. Bet kuriuo atveju, darbas naudingas visiems dirbantiems su WEB aplikacijomis, kaip informacijos ir naujų idėjų šaltinis.

Literatūra

[1] The Other Road Ahead
Paul Graham, 2001

[2] MVC
XEROX PARC, 1978-79

[3] Building user interfaces for object-oriented systems, Part 1
Allen Holub, 1999

[4] Executable UML: A Foundation for Model-Driven Architecture
Stephen J. Mellor, Marc J. Balcer, 2002

[5] Patterns and Software: Essential Concepts and Terminology
Brad Appleton, 2000

[6] Component-Oriented Programming Languages: Messages vs. Methods, Modules vs. Types
Peter H. Fröhlich, 2000

[7] Visual Programming Languages: A Survey
Marat Boshernitsan, Michael Downes, 1997, University of California, Berkeley

[8] Patterns of Visual Programming
Dave Parsons, Mark Cranshaw, 2001

[9] How To Break Web Software - A look at security vulnerabilities in web software
Mike Andrews, 2006, Google TechTalks

Nuorodos

[10] cantata: A Data Flow Visual Language
http://greta.cs.ioc.ee/ khoros2/k2tools/cantata/cantata.html

[11] XML Schema Part 0: Primer Second Edition
http://www.w3.org/TR/xmlschema-0/

[12] Dia a drawing program
http://www.gnome.org/projects/dia/

[13] Qt – Trolltech
http://www.trolltech.com/products/qt

[14] Šiauliai Mathematical Seminar
http://dim.su.lt/ matkat/ (laikina nuoroda)