KAUNO TECHNOLOGIJOS UNIVERSITETAS
INFORMATIKOS FAKULTETAS
PROGRAMŲ INŽINERIJOS KATEDRA

Vytautas Barkauskas

UML MODELIŲ PRAPLĖTIMO OCL APRIBOJIMAIS GALIMYBIŲ TYRIMAS

Magistro darbas

Darbo vadovas
prof. Eduardas Bareiša

Kaunas, 2009

Turinys

Turinys....2
SUMMARY....3
Paveikslai....4
Įvadas....5
1.1.Dokumento paskirtis....5
1.2.Santrauka....5
2.UML modeliavimo įrankio analitinė dalis....6
2.1.UML ir susijusių standartų analizė....6
UML standartas....6
2.1.1.OCL standartas....9
2.2.Egzistuojančių UML modeliavimo įrankių analizė....10
2.2.1.UML modeliavimo įrankiai....10
2.2.2.Įrankių vertinimo kriterijai....11
2.2.3.Verslo galimybės....13
2.2.4.Projektavimo metodologijos ir technologijų analizė....13
3.Projektinė dalis....16
3.1.Sistemos funkciniai reikalavimai....16
3.1.1.Redaktoriaus funkciniai reikalavimai....16
3.1.2.UML įskiepio funkciniai reikalavimai....19
3.2.Sistemos architektūros pateikimas....20
3.3.Architektūros tikslai ir apribojimai....20
3.4.Sistemos statinis vaizdas....21
3.5.Architektūros kokybė....24
4.Tyrimo dalis....25
4.1.OCL išraiškomis paremtas metamodelio objektų validavimas....28
5.Eksperimentinė dalis....29
5.1.Automatinė OCL išraiškomis paremta validavimo sistema....29
5.1.1.Realizavimo ypatumai....30
5.1.2.Bandymai....30
5.1.3.Eksperimento išvados....31
5.2.Kompiliuotų ir generuotų elementų spartos palyginimas....31
5.2.1.Testai....31
5.2.2.Bandymų aplinkos....32
5.2.3.Bandymų rezultatai....33
5.2.4.Bandymų išvados....34
6.Išvados....35
7.Literatūra....36
8.Terminų ir santrumpų žodynas....37
9.Priedai....38
9.1.Įrankyje naudota OCL paremta gramatika....38

SUMMARY

This document describes the work that consists of four main parts. Analysis of UML and related modeling standards was performed in the first – analytical – part. UML editors that are popular in the present market and their advantages were reviewed. The possibilities of making business in this area were discussed.

The functionality, architectural solutions and architecture quality criteria of the developed system were reviewed in the second part of the work.

Researches of possibilities of code and library generating from UML models are described in the third part of the work. The proposal to apply OCL expressions for element property validation was offered.

The substantiation of realization of the system for element property validation by OCL restrictions is described in the last part of the work. The experiment that compares characteristics of the compiled dynamic class libraries speed and the generated dynamic class libraries speed is presented.

Paveikslai

Paveikslas 1. UML 2 diagramų klasifikavimas.....7
Paveikslas 2. Eksperimento dalyvių atsiliepimas apie UML prieš eksperimentą [7].....8
Paveikslas 3. Eksperimento dalyvių atsiliepimas apie UML po eksperimento [7].....9
Paveikslas 4. Modelių redaktoriaus panaudos atvejai.....16
Paveikslas 5. UML įskiepio panaudos atvejai.....19
Paveikslas 6. Sistemos komponentų diagrama.....21
Paveikslas 7. Sistemos paketų diagrama.....22
Paveikslas 8. UML specifikacijose pateikiamo daugybinio paveldėjimo pavyzdys.....25
Paveikslas 9. UML modelio elementų vaizdas atlikus transformacijas.....26
Paveikslas 10. MDA modelių transformacijos.....27
Paveikslas 11. Validuojamų elementų tipų žymėjimas.....30
Paveikslas 12. Elementų savybių reikšmių validavimo iškvietimas.....30
Paveikslas 13. Pirmosios testavimo sistemos rezultatai 33
Paveikslas 14. Antrosios testavimo sistemos rezultatai.....34

1. Įvadas

Šio magistro baigiamojo darbo tikslas yra sukurti UML modeliavimo įrankį, palaikantį OCL apribojimus, ir ištirti UML ir OCL kombinacijos taikymo galimybes. Projekto vystymo eigoje atsirado poreikis generuoti didelius klasių kiekius, todėl pats darbas vystytas dinaminių klasių bibliotekų generavimo tyrimo linkme. Darbe validavimui pritaikyta OCL išraiškų ir apribojimų kalba.

1.1. Dokumento paskirtis

Šis dokumentas yra programų inžinerijos magistro baigiamasis darbas. Dokumente apžvelgėme UML modeliavimo įrankių rinką, inžinerijai svarbias jų savybes. Aprašėme UML redaktoriaus kūrimo procesą, bei specifines ir technines proceso subtilybes.

Dokumente pateikėme architektūrinius sprendimus, kurie buvo priimti projektuojant sistemą. Taip pat buvo pateikti generuotos ir kompiliuotos klasių bibliotekų palyginimo rezultatai dviejuose skirtingose testavimo aplinkose.

1.2. Santrauka

Šiame dokumente aprašytas darbas susideda iš keturių pagrindinių dalių. Pirmojoje, analitinėje dalyje, atlikome UML ir susijusių modeliavimo standartų analizę. Apžvelgėme populiariausius rinkoje egzistuojančius UML redaktorius ir jų stipriąsias puses. Aptarėme verslo šioje rinkoje galimybes.

Antrojoje darbo dalyje apžvelgėme sukurtos sistemos funkcionalumą, architektūrinius sprendimus ir architektūros kokybinius kriterijus.

Trečioje dalyje aprašomi kodo ir bibliotekų generavimo iš UML modelių galimybių tyrimai. Pateiktas siūlymas taikyti OCL išraiškas elementų savybių validavimui.

Paskutinėje dalyje aprašomas sistemos, skirtos elementų savybių validavimo OCL apribojimų pagalba, realizacijos pagrindimas, pateikia. Taip pat pateikiamas eksperimentas, palyginantis sukompiliuotų ir sugeneruotų dinaminių klasių bibliotekų spartos charakteristikas.

2. UML modeliavimo įrankio analitinė dalis

2.1. UML ir susijusių standartų analizė

UML standartas

UML yra grafinė modeliavimo kalba, kurti objektiškai orientuotų sistemų specifikacijas, modeliuoti jų architektūrą ir veikimo terpę. UML jau tapo de-fakto standartu programų sistemų modeliavime. Šio standarto vystymą palaiko Objektų valdymo grupė OMG. Naujausia UML specifikacijos versija 2.2 aprašo per 200 modelio elementų ir 13 diagramų [7] suskirstytų į 3 kategorijas:

Struktūros diagramos

Struktūros diagramos išskiria kokie elementai turi egzistuoti modeliuojamoje sistemoje.

Išdėstymo (Deployment) diagramos tipas skirtas modeliuoti fizinį sistemos elementų išdėstymą.

Klasių (Class) – aprašo sistemos struktūrą, atvaizduodama sistemos klases, kasių atributus, operacijas ir ryšius tarp klasių.

Komponentų (Component) – atvaizduoja kaip programų sistema išskaidyta į komponentus ir nurodo jų tarpusavio priklausomybes.

Kompozicinės struktūros (Composite structure) – aprašo vidinę klasių struktūrą ir struktūros suteikiamas bendradarbiavimo galimybes.

Objektų (Object) – įgalina sudaryti pilną arba dalinį modeliuojamos sistemos vaizdą konkrečiu pegeidaujamu aprašyti momentu.

Paketų (Package) – atvaizduoja kaip sistema suskaidyta į logines grupes, skirdama pagrindinį dėmesį priklausomybėms tarp šių grupių.

Elgsenos diagramos

Ši diagramų grupė akcentuoja kas turi vykti modeliuojamoje sistemoje.

Būsenų mechanizmo (State Machine) – standartizuotos notacijos pagalba suteikia galimybę apibrėžti konkretaus elemento būsenų keitimo mechanizmus.

Panaudos atvejų (Use Case) – atvaizduoja vartotojams (aktoriams) teikiamą sistemos funkcionalumą (panaudos atvejus). Aktorių tikslai atvaizduojami kaip panaudos atvejai ir jų tarpusavio ryšiai.

Veiklų (Activity) – leidžia atvaizduoti pažingsninę sistemos komponentų veiklų seką.

Sąveikos diagramos

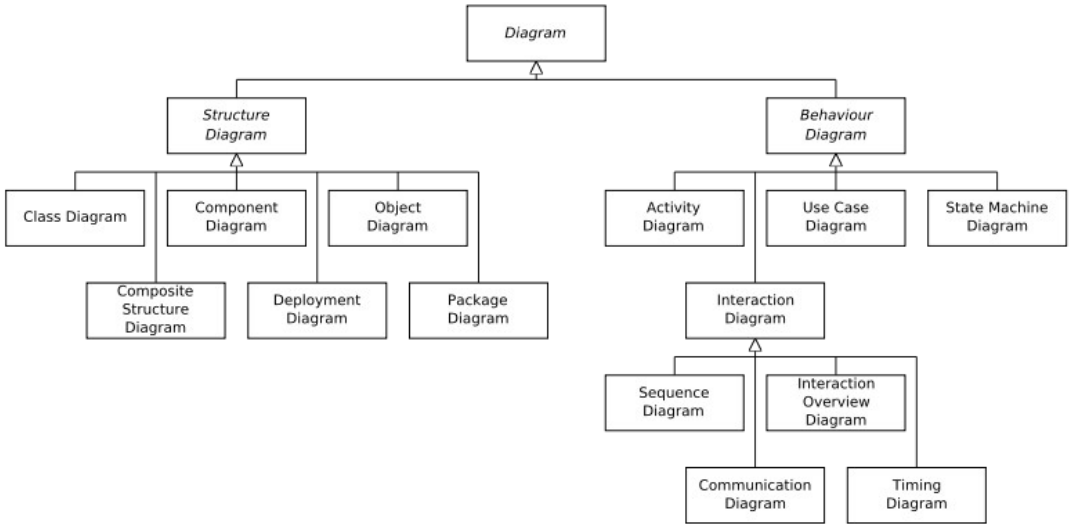
Sąveikos diagramos – tai elgsenos diagramų grupės pogrupis, akcentuojantis į modeliuojamos sistemos elementų duomenis ir valdymo sekas.

Komunikacijų (Communication) – nuoseklių pranešimų pagalba atvaizduoja sąveikas tarp objektų ar jų dalių. Šis diagramų tipas apjungia informaciją paimtą iš klasių, sekų ir panaudos atvejų diagramų ir apjungia statinę ir dinaminę sistemos elgseną.

Laikinio pasiskirstymo (Timing) – specifinis sąveikos diagramų tipas, kuriame esminis akcentuojamas dalykas yra laikiniai apribojimai.

Sąveikos apžvalgos (Interaction Overview) – tai atskiras veiklų diagramų atvejis, kuriame viršūnės atitinka sąveikos diagramas.

Sekų (Sequence) – šiose diagramose atvaizduojama, kaip objektai bendradarbiauja pranešimų sekomis. Šios sekos nurodo objektų gyvavimo trukmę.

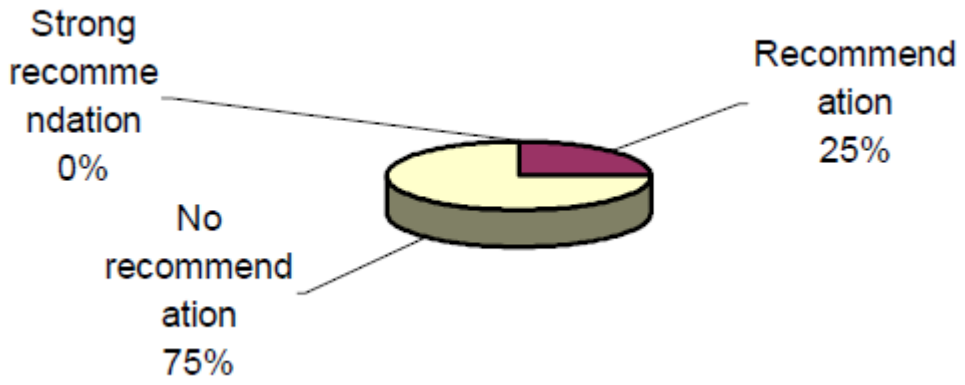


Paveikslas 1. UML 2 diagramų klasifikavimas

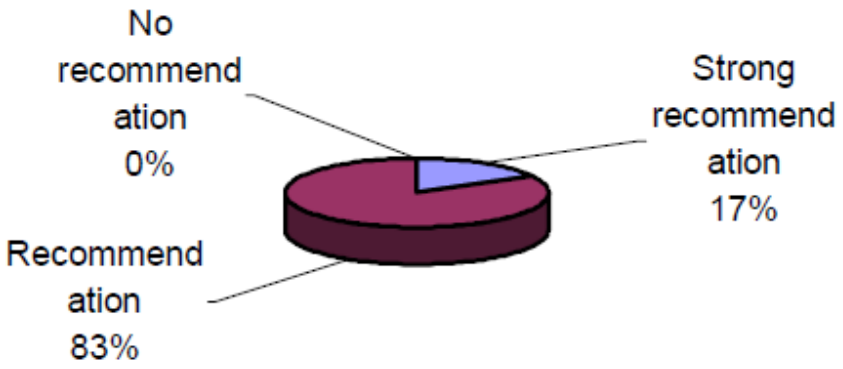
UML taikymas

UML paskirtis – programinės įrangos specifikavimas modulio sudarymas. UML kalbos populiarumas ir įrankių gausa skatina naujų taikymo būdų paiešką. Daugelis populiariausių komercinių UML CASE įrankių palaiko dokumentacijos ir kodo šablonų generavimą populiariausioms programavimo kalboms, tokioms kaip Java, C++, C#, PHP. Dalis jų turi ir atvirkštinės inžinerijos priemonių, galinčių sudaryti sistemos modelį iš išeities tekstų.

Nors UML standartas ne retai sulaukia ir kritikos [7], tačiau jo paplitimas vis auga. Vladimir Pavlov 2005 metais ACM simpoziume pristatė eksperimento „Babelio eksperimentas: pažangus pantomima paremtas objektiškai orientuotos analizės ir projektavimo mokymas su UML“ rezultatus. Eksperimento tikslas buvo „leisti eksperimento dalyviams atrasti ar UML yra „reali kalba“ tinkanti ir naudinga projektą realizuojančiai komandai“[7]. Eksperimento metu dalyviai negalėjo bendrauti nei žodžiu nei raštu, tik per UML. Eksperimentas pasitvirtino daugiau nei dešimtį kartų tiek akademinėse tiek korporacinėse aplinkose ir sukėlė teigiamą studentų ir klientų atsakomąją reakciją“[7].



Paveikslas 2. Eksperimento dalyvių atsiliepimas apie UML prieš eksperimentą [7]



Paveikslas 3. Eksperimento dalyvių atsiliepimas apie UML po eksperimento [7]

2.1.1. OCL standartas

OCL yra formali kalba, skirta aprašyti išraiškoms UML modeliuose. Šios išraiškos dažniausiai būna invariantinės sąlygos, kurias turi tenkinti modeliuojama sistema arba užklausos, taikomos modelyje aprašomiems objektams. OCL specifikacija pabrėžia, kad šių išraiškų vykdymas negali sukelti šalutinio poveikio sistemoje, t.y. jų tikrinimas negali niekaip pakeisti vykdomos sistemos būsenos [7].

OCL specifikacija nurodo šiuos panaudojimo atvejus:

- taikyti kaip užklausų kalbą;
- specifikuoti invariantus modeliuojamos sistemos klasėse ir tipuose;
- specifikuoti invariantus stereotipams;
- aprašyti prieš ir po sąlygas operacijoms ir metodams;
- aprašyti saugams (Guards) pereinant tarp būsenų ar veiklų;
- aprašyti tikslo elementus pranešimams(Messages) ir veiksmams(Actions);
- aprašyti operacijų apribojimus;
- aprašyti išvedimo taisykles atributams bet kokiai išraiškai UML modelyje.

2.2. Egzistuojančių UML modeliavimo įrankių analizė

UML yra labiausiai paplitęs programinės įrangos specifikavimo ir architektūros projektavimo standartas, todėl šiuo metu galime skaičiuoti dešimtis tiek mokamų tiek nemokamų grafinių UML redaktorių. Šiame skyriuje apžvelgsime tik kelis plačiausiai žinomus ir labiausiai išbaigtus įrankius.

2.2.1. UML modeliavimo įrankiai

MagicDraw UML

Šis vizualinio UML modeliavimo įrankis kuriamas Lietuvoje. Naujausia įrankio versija palaiko UML 2.2, SysML 1.1, OCL 2.0 modeliavimo standartus. Senesnės versijos palaiko ir UML 1.4 standartą. Turi pilną tiesioginės ir atvirkštinės inžinerijos funkcionalumą Java, J2EE, C#, C++, CORBA IDL ir SQL kalboms. Įrankis gavęs apdovanojimų už produktyvumą ir kaip geriausias modeliavimo įrankis.

Rational Software Architect

Rational Software Architect projektavimo ir programinės įrangos kūrimo įrankis skirtas aukšto lygio programinių paslaugų ir modeliais paremtų (MDD) aplikacijų kūrimui. RSA įrankis sukurtas kaip Eclipse programų kūrimo aplinkos įskiepis, todėl suteikia galimybę viename redaktoriuje kurti modelius ir rašyti programinį kodą. Naujausia įrankio versija palaiko UML 2.1 standartą. RSA turi tiesioginės inžinerijos priemones Java, C#, C++, EJB, WSDL, XSD, CORBA IDL, SQL kodo generavimui ir gali sudaryti kuriamos sistemos modelį iš Java ir C++ programinės įrangos kodo. Įrankis taipogi turi kodo analizės ir keitimo (angl. refactoring), bei komandinio darbo priemones.

Enterprise Architect

Enterprise Architect 7.5 yra modeliavimo ir vizualinio redagavimo platforma, paremta UML 2.1 ir susijusiais standartais. Be UML platforma palaiko sisteminio modeliavimo kalbą SysML ir biznio procesų modeliavimo notaciją BPMN. Reikalavimų dokumentavimo ir valdymo priemonės leidžia generuoti modeliuojamos sistemos dokumentaciją. Įrankis turi komandinio darbo galimybes, palaiko tiesioginę bei atvirkštinę

inžineriją (angl., round-trip engineering) ActionScript, C#, C++, Corba IDL, Delphi, Java, PHP, Python, Visual Basic ir Visual Basic.NET programavimo kalboms.

Together

Borland kompanijos modeliavimo įrankis Together kaip įskiepis integruojamas į Eclipse (Java versija) arba Microsoft Visual Studio (.NET versija) programavimo aplinkas. Palaiko UML 2.0 ir UML 1.4 versijas. Turi priemones biznio procesų modeliavimui BPMN notacija. Įrankis palaiko abipusę automatinę modelis–kodas sinchronizaciją ir turi statinio kodo analizės funkcijas.

Visual Paradigm

Kaip ir dauguma analizuotų įrankių palaiko UML 2.1 ir visų 13 tipų diagramas. Kaip ir kiti tirti įrankiai palaiko tiesioginės ir atvirkštinės inžinerijos metodus Java, C++, Corba IDL kalboms, turi komandinio darbo galimybes. Deja neturi jokio OCL palaikymo.

2.2.2. Įrankių vertinimo kriterijai

UML 2 palaikymas – geras UML įrankis privalo palaikyti naujausius modeliavimo standartus. Šiuo metu nėra nei vieno įrankio, kurio kūrėjai skelbtų pilnai realizavę visas UML 2 funkcijas ir galimybes, tačiau dauguma įrankių leidžia kurti visų tipų diagramas ir jose panaudoti visus ar didžiąją dalį UML 2 elementų.

UML 1.4 palaikymas – ši UML versija išskirta, nes kol kas ji vienintelė tarptautinės standartų organizacijos ISO pripažinta kaip standartas (ISO/IEC 19501:2005).

XMI 2.0 palaikymas – XMI yra OMG grupės standartas skirtas su MOF suderinamų metaduomenų apsikeitimui. Šis standartas suteikia galimybę skirtingiems įrankiams apsikeisti UML modeliais.

Tiesioginė inžinerija – kodo generavimas iš UML modelių bent populiariausioms programavimo kalboms, tokioms kaip Java, C++, C#, šiuolaikiniuose modeliavimo įrankiuose yra būtinybė. Paketų, komponentų, klasių, atributų it operacijų šablonų generavimas taupo sistemos kūrėjų laiką.

Atvirkštinė inžinerija – atvirkštinės inžinerijos galimybės naudingos norint gauti UML modelį iš turimo programinės įrangos kodo. Modernūs įrankiai iš kodo gali išgauti paketus, klases, atributus, operacijas, ryšius tarp klasių ir kitokią naudingą informaciją. Gudresni įrankiai specifiniais atvejais tokią informaciją gali išgauti ir iš vykdomųjų programų ar bibliotekų.

Dokumentavimo galimybės – UML standartuose numatyta galimybė bet kokio tipo elementus susieti su komentarais, tačiau šiuolaikiškų UML įrankių kūrėjai jas dar labiau išplečia suteikdami galimybę kurti specifinius elementų aprašus. Šių aprašų informacija praverčia generuojant kodą, arba specifines ataskaitas.

OCL palaikymas – šis funkcionalumas leidžia UML modelius papildyti OCL išraiškom. Pačios išraiškos gali būti naudojamos modelio validavimui, atributų išvedimo taisyklėm aprašyti ir metodų veikimo specifikavimui.

OCL interpretavimas – interpretavimas suteikia galimybę aptikti klaidas OCL išraiškose, paryškinti sintaksę ir yra labai esminis komponentas norint plačiau taikyti OCL.

Modelių validavimas OCL pagalba – šis funkcionalumas leidžia apibrėžti apribojimus pačių modelių sudarymui. Taip galima modeliuojamoje sistemoje įvesti daugiau aiškumo ir patikrinti ar modeliai atitinka jiems keliamus reikalavimus.

Lentelė 1. UML įrankių palyginimas

Savybės / Įrankiai	MagicDraw UML	Rational Software Architect	Enterprise Architect	Together	Visual Paradigm
UML 2 palaikymas	*	*	*	*	*
UML 1.4 palaikymas	*	*	*	*	*
XMI 2.0 palaikymas	*	*	*	*	*
Tiesioginė inžinerija	*	*	*	*	*
Atvirkštinė inžinerija	*	*	*	*	*
Dokumentavimo galimybės	*	*	*	*	*
OCL palaikymas	*	*	*	*	
OCL interpretavimas	*	*	*	*	
Modelių validavimas OCL	*	*	*	*	

2.2.3. Verslo galimybės

Šiai dienai yra sukurta dešimtys mokamų ir nemokamų UML modeliavimo įrankių, todėl verslo galimybės šioje rinkoje pakankamai ribotos. Dauguma įrankių be UML palaiko ir kitokius modeliavimo standartus bei notacijas. Naujo komercinio bendros paskirties UML įrankio pateikimas rinkai nieko nenustebintų ir nesudomintų. Norint gauti finansinės naudos reiktų įrankį papildyti naujomis savybėmis, kurių kiti įrankiai neturi. Deja numatyti kokių papildomų naujų funkcijų gali pageidauti vartotojas, neskyrus tam papildomų lėšų, nėra lengva.

Vienas iš įrankio privalumų tas, kad jis realizuotas .NET karkaso pagalba, todėl gali pilnai išnaudoti .NET teikiamas galimybes, tokias kaip .NET ansamblių (angl. assembly) refleksija. Dauguma UML įrankių parašyti Java arba C++. Patobulinus redaktorių yra reali, bet maža tikimybė, kad atsirastų norinčių jį įsigyti.

2.2.4. Projektavimo metodologijos ir technologijų analizė

Tiesioginė inžinerija

Sistemų apimtys laikui bėgant vis didėja ir jų sudėtingumas nuolat auga, todėl natūralu ieškoti būdų kaip sistemų kūrimo procesą kuo daugiau automatizuoti. Tiesioginė inžinerija, realizuojama per kodo generavimą, gali padėti išvengti didelės dalies kodavimo darbų.

CIL

CIL (angl. Common Intermediate Language) yra žemiausio lygio programavimo kalba .NET karkaso bendros kalbos infrastruktūroje (angl. Common Language Infrastructure). Kalba yra objektiškai orientuota ir paremta steku. Šios kalbos instrukcijas vykdo virtuali mašina. Visos .NET karkasą palaikančiomis programavimo kalbomis parašytos programos kompiliuojamos į CIL kodą. CIL sudaryta iš nei nuo procesoriaus, nei nuo platformos nepriklausomos instrukcijų aibės, todėl jos kodas gali būti vykdomas bet kokioje aplinkoje, palaikančioje .NET karkasą.

Šiame projekte CIL instrukcijos naudojamos UML modelio elementų generavimui į dinamines bibliotekas (DLL). Sugeneruotos bibliotekos užima mažiau vietos ir yra greičiau vykdomos.

Refleksija

Kompiuterių moksle refleksija (angl. reflection) vadinamas procesas, kurio metu kompiuterinė programa gali stebėti ir modifikuoti savo struktūrą ir elgseną. Tai viena iš metaprogramavimo rūšių. Refleksijos pagalba galima išgauti informaciją apie objekto tipą, jam priklausančius laukus ar metodus, realizuojamus interfeisus.

XML

Tai bendros paskirties duomenų struktūrų bei jų turinio aprašymo kalba. Pagrindinė XML paskirtis yra užtikrinti lengvesni apsikeitimą duomenimis tarp skirtingų sistemų. XML dokumentas sudarytas iš elementų, kurie visada turi vardą ir gali turėti neribotą kiekį atributų. Elemento viduje gali būti įsuptas tekstas, arba kiti, jam priklausantys, elementai. Viena pagrindinių XML privalumų yra tai, kad XML informaciją saugo žmogui lengvai suprantamu tekstiniu formatu. Projekte XML naudojamas konfigūracijos saugojimui.

XMI

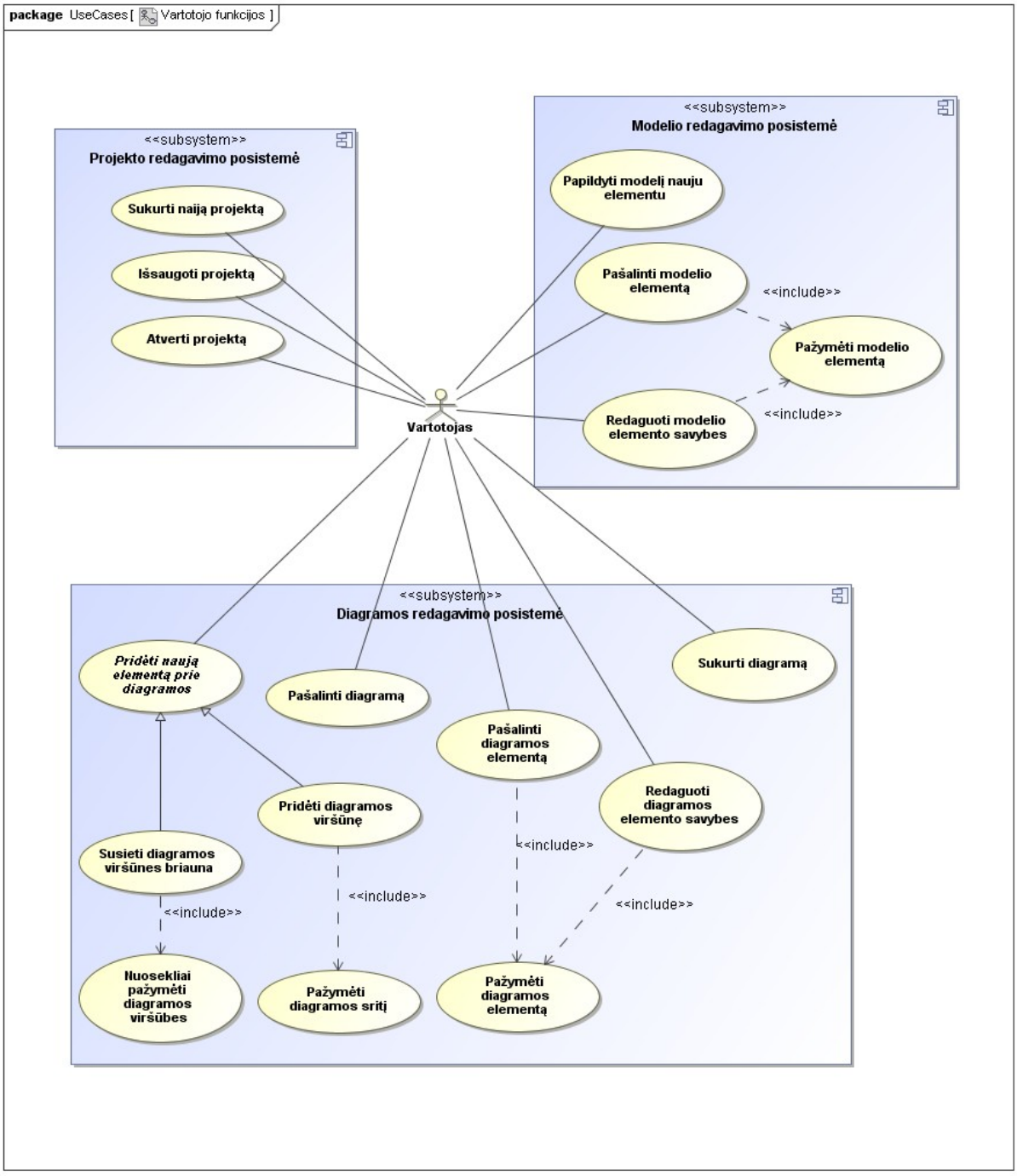
Tai standartas metaduomenų apsikeitimui naudojantis XML. Gali būti naudojamas bet kokių metaduomenų, kurių metamodelis paremtas MOF standartu, apsikeitimui, tačiau dažniausiai taikomas UML. Šis standartas aprašo, kaip metamodelio elementai ir jų duomenys aukštesniame abstrakcijos lygyje gali būti saugomi XML duomenų struktūromis.

3. Projektinė dalis

3.1. Sistemos funkciniai reikalavimai

3.1.1. Redaktoriaus funkciniai reikalavimai

Sistemos funkcinių reikalavimų specifikacija, perteikta panaudos atvejų diagrama.



Paveikslas 4. Modelių redaktoriaus panaudos atvejai

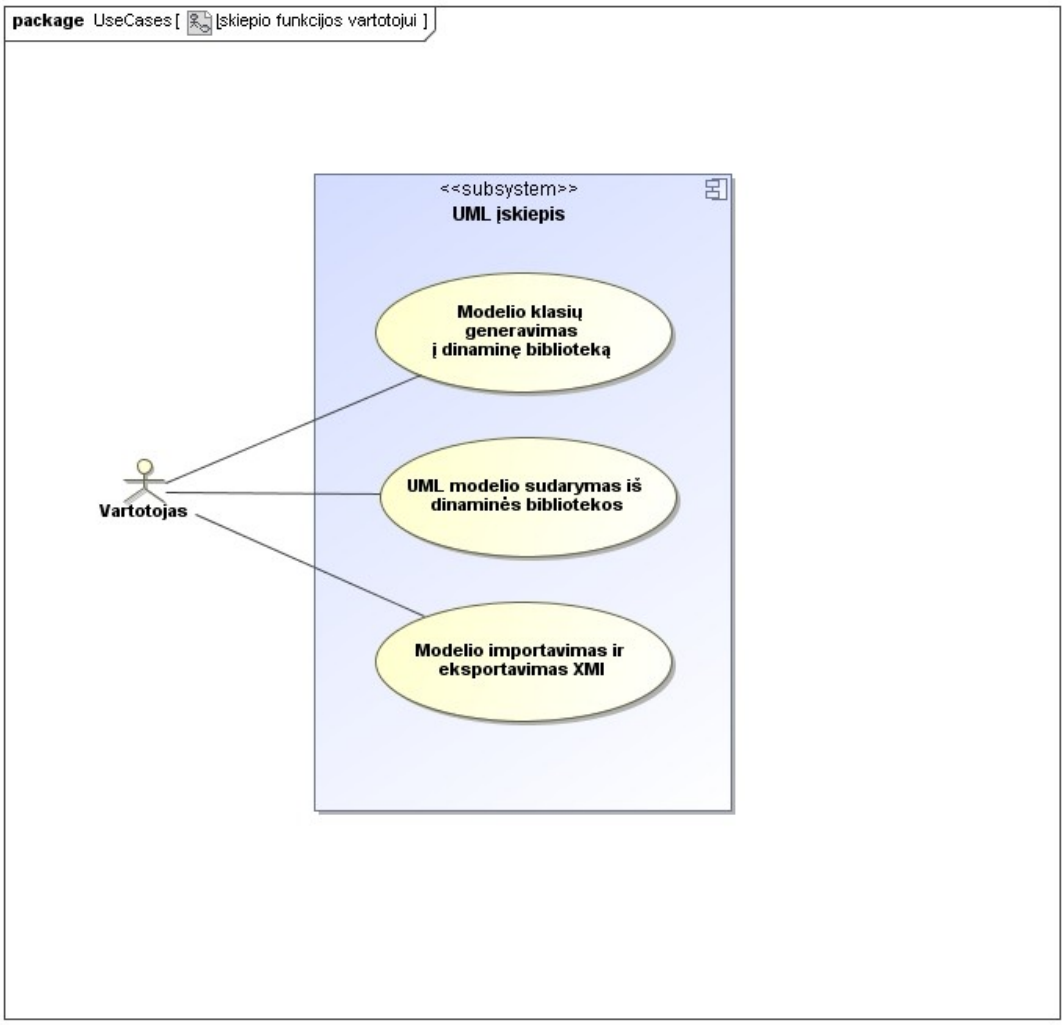
Sistemos vartotojui teikiami panaudos atvejai

- **Projekto redagavimo posistemė** – ši posistemė apjungia projekto kūrimo ir redagavimo funkcijas:
    - **Sukurti naują projektą** – Vartotojas gali sukurti naują projektą. Projekto kūrimą realizuoja įskiepiai. Įskiepis gali suteikti vartotojui galimybę sukurti projektą pasirenkant iš anksto paruoštų šablonų.
    - **Išsaugoti projektą** – Sistema suteikia vartotojui galimybę išsaugoti sukurtą ar atvertą projektą. Sistema neriboja projekto išsaugojimo vien tik projektą realizuojančiam įskiepiui.
    - **Atverti projektą** – sistema realizuoja funkcionalumą, reikalingą atverti ir užkrauti išsaugotam projektui.
- **Modelio redagavimo posistemė** – ši posistemė apima modelio redagavimo funkcijas:
    - **Papildyti modelį nauju elementu** – vartotojas gali papildyti modelį konkrečiam projekto tipui numatytais elementais. Projekto valdymo priemonės gali riboti kur ir kokius elementus galima pridėti.
    - **Pašalinti modelio elementą** – naudotojas gali pašalinti bet kurį, išskyrus aukščiausio lygio, modelio elementą. Šalinant elementą, rekursiškai panaikinami ir visi jame laikomi elementai ir su juo susiję ryšiai. Kartu pašalinami ir modelio elementus diagramose atvaizduojantys elementai.
    - **Redaguoti modelio elemento savybes** – sistema užtikrina modelio elementų savybių redagavimą.
    - **Pažymėti modelio elementą** – modelio pažymėjimas leidžia vykdyti tolesnius modelio redagavimo veiksmus – elemento šalinimą ar jo savybių redagavimą.
- **Diagramos redagavimo posistemė** – ši posistemė apima su diagramų redagavimu susijusias sistemos funkcijas:
    - **Sukurti diagramą** – vartotojas gali papildyti projektą konkrečiam projekto tipui numatytomis diagramomis.
    - **Pridėti naują diagramos elementą** – abstraktus panaudos atvejis. Diagramos sandara interpretuojama kaip grafas t.y. yra sudaryta iš viršūnių ir briaunų. Šis panaudos atvejis realizuojamas per šias konkrečias realizacijas:

        - **Pridėti naują diagramos viršūnę** – pasirinkęs norimą elementą ir pelės pagalba apibrėžęs pageidaujamo dydžio sritį diagramoje, vartotojas gali įkelti viršūnės elementą.
        - **Susieti diagramos viršūnes briauna** – pasirinkęs pageidaujamą ryšio elementą ir nurodęs pageidaujamas sujungti viršūnes vartotojas gali susieti diagramos viršūnes briauna.
    - **Pažymėti diagramos sritį** – diagramos srities apibrėžimas pele, priklausomai nuo vykdomų veiklų, gali pažymėti srityje esančius elementus arba nurodyti pradinius matmenis įkeliamai diagramos viršūnei.
    - **Nuosekliai pažymėti diagramos viršūnes** – diagramos viršūnių susiejimo briaunomis funkcijai realizuoti reikalinga nuoseklaus diagramos viršūnių pažymėjimo funkcija. Projekto valdiklių realizacija gali riboti konkrečių briaunų ir viršūnių sujungimą.
    - **Redaguoti diagramos elemento savybes** – sistema leidžia vartotojui keisti vizualines diagramos elementų savybes.
    - **Pašalinti diagramos elementą** – šalinant diagramos elementą panaikinami ir visi nuo jo egzistavimo priklausomi elementai.
    - **Pažymėti diagramos elementą** – ši funkcija įgalina tolimesnį pažymėtų diagramos elementų redagavimą.
    - **Pašalinti diagramą** – vartotojas gali pašalinti nepageidaujamas diagramas iš projekto.

3.1.2. UML įskiepio funkciniai reikalavimai

UML įskiepio funkciniai reikalavimai pateikti panaudos atvejų diagrama.



Paveikslas 5. UML įskiepio panaudos atvejai

UML įskiepio vartotojui teikiami panaudos atvejai

- **Modelio klasių generavimas į dinaminę biblioteką** – parinkdamas pageidaujamus vartotojas gali sugeneruoti dinaminę .NET biblioteką.
- **UML modelio sudarymas iš dinaminės bibliotekos** – vartotojas gali importuoti modelį iš .NET dinaminės bibliotekos.
- **Modelio importavimas ir eksportavimas XMI** – vartotojas gali UML modelį importuoti iš ir eksportuoti į XMI standarto bylą.

3.2. Sistemos architektūros pateikimas

Projekto dokumentacija buvo sudaryta naudojantis „No Magic, Inc.“ įrankiu „MagicDraw“. Visos architektūros specifikavimui naudota UML 2.0 notacija ir diagramos.

Paketų išdėstymo diagramoje reiktų atkreipti dėmesį kad paketas šiuo atveju yra jame talpinamų elementų vardų sritis, o paketai pažymėti UML stereotipu „assembly“ yra atskiri vykdomieji vienetai, t.y. dinaminės bibliotekos ar vykdomosios bylos.

3.3. Architektūros tikslai ir apribojimai

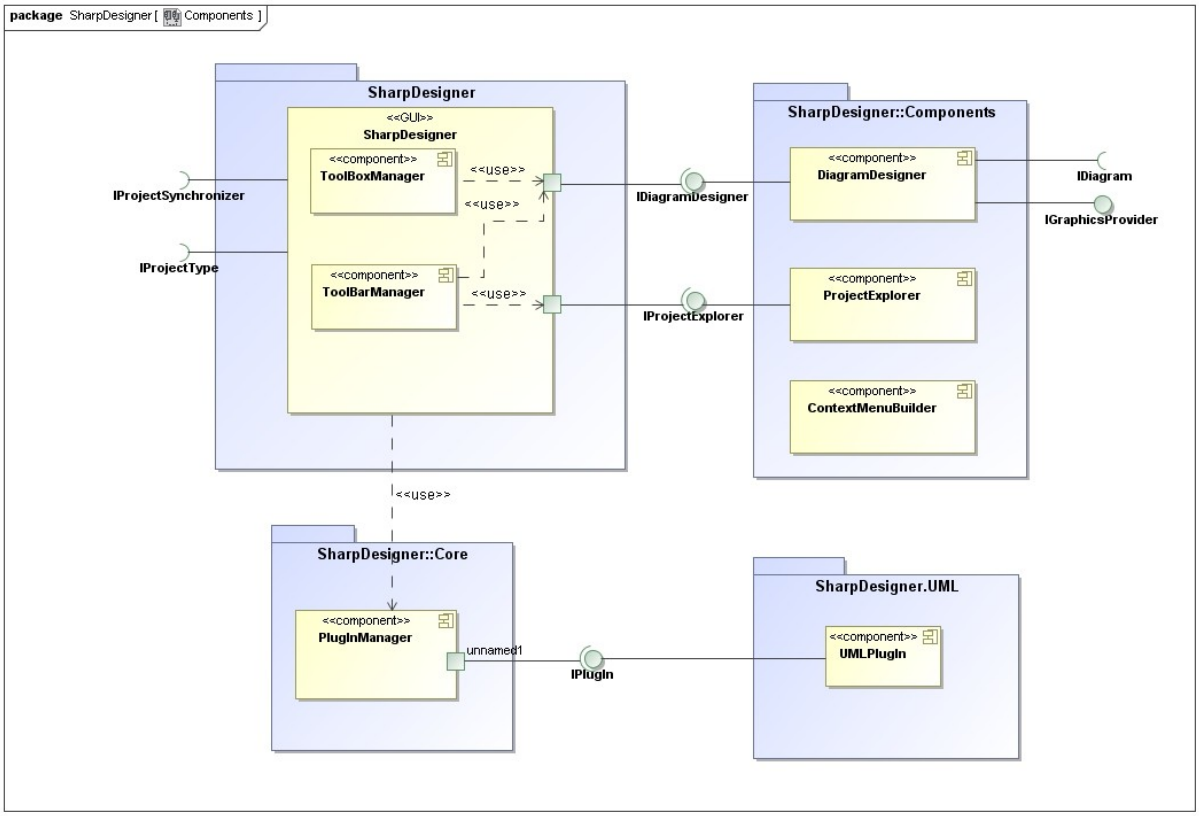
Iš anksto buvo žinoma, kad kuriama sistema bus realizuojama naudojant .NET 2.0 karkasą, kuris yra skirtas Windows operacinių sistemų šeimai. Egzistuoja projektas Mono, kuris dalinai realizuoja .NET karkasą Linux operacinėms sistemoms. Prieš pradedant įgyvendinti projekto programinę realizaciją buvo atliekama analizė skirtumams tarp Windows .NET ir Linux Mono karkasų. Egzistuojanti skirtumai pasirodė per dideli kad sistemą realizuoti abiem karkasam, todėl sistema veiks tik Windows operacinėje sistemoje.

Projekto įgyvendinimą finansiškai riboja asmeninės lėšos, todėl jokių komercinių komponentų naudojama nebus. Sistemos sąsają sudarys .NET karkaso teikiami ir iš jų išvesti komponentai.

3.4. Sistemos statinis vaizdas

Sistemos realizacijoje grafinio redagavimo, modelio elementų interfeisų, bei jų bazinių realizacijų klasės išskaidytos į atskirus modulius. Modeliavimo karkaso

Esminiai architektūros aspektai pateikti komponentų diagramos pagalba.



Paveikslas 6. Sistemos komponentų diagrama

Pagrindiniai sistemos komponentai:

DiagramDesigner – diagramų redagavimą komponentas įgalinantis. Kiekviena diagrama gali turėti skirtingus redagavimo metodus.

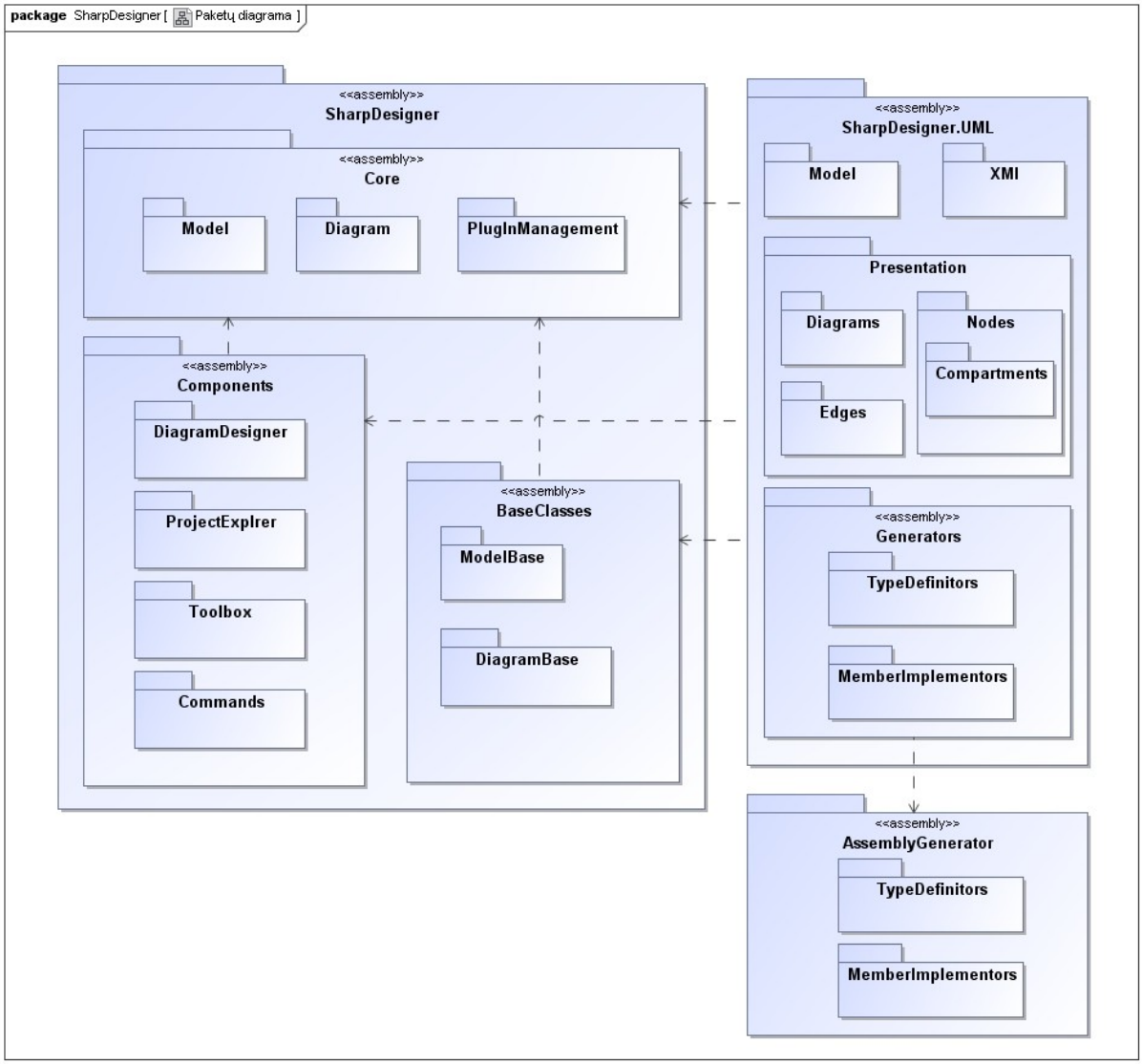
ProjectExplorer – redaguojamo modelio medžio atvaizdavimą bei su tuo susijusias funkcijas valdantis komponentas.

ContextMenuBuilder – komponentas atsakingas už kontekstinių meniu generavimą ir atvaizdavimą.

ToolBoxManager – komponentas, pagal diagramos tipą, atrenkantis leidžiamus naudoti elementus ir sukuriantis įrankių juostos komponentus.

ToolBarManager – komponentas valdantis papildomus modelio ir diagramų redagavimo metodus.

PlugInManager – šis komponentas atsakingas už įskiepių valdymą.



Paveikslas 7. Sistemos paketų diagrama

- **SharpDesigner** – pagrindinis sistemos paketas, jame talpinami visi kiti sistemai priklausantys paketai ir elementai.
    - **Core** – šiame pakete saugomi pagrindiniai interfeisai ir klasės. Jie sudaro sistemos branduolį.
        - **Model** – interfeisai aprašantys modelio elementus.
        - **Diagram** – interfeisai aprašantys diagramas ir jų elementus.
        - **PlugInManagement** – įskiepių interfeisai ir klasės darbui su jais.
    - **Components** – šiame pakete laikomi grafiniai sistemos komponentai.
        - **DiagramDesigner** – diagramų redagavimo komponentas ir pagalbinės klasės bei interfeisai.
        - **ProjectExplorer** – projekto atvaizdavimo medžio struktūra komponentas ir pagalbinės klasės bei interfeisai.
        - **ToolBox** – įrankių juostų komponentai ir pagalbinės klasės bei interfeisai.
        - **Commands** – bazinės klasės diagramų ir modelio redagavimo funkcionalumui realizuoti.
    - **BaseClasses** – šis paketas skirtas įskiepių kūrimo palengvinimui. Jame laikomos bazinės klasės realizuojančios modelio ir diagramų elementus.
        - **ModelBase** – bazinės klasės modelio kūrimui.
        - **DiagramBase** – bazinės klasės diagramų kūrimui.
    - **UML** – įskiepio paketas kuriame talpinami UML modelio ir diagramų elementai.
        - **Model** – paketas kuriame laikomi UML modeliai.
        - **Presentation** – šiame pakete talpinami UML grafiniai elementai,
            - **Diagrams** – pakete laikomos UML diagramos.
            - **Nodes** – šiame pakete talpinamos UML diagramų viršūnės, t.y. diagramų objektai.
                - **Compartments** – diagramos elementų sudedamosios dalys kurios naudojamos keliuose skirtingų tipų diagramų objektuose.
            - **Edges** – pakete laikomos diagramų briaunos, arba kitaip – ryšiai.

    Generators – paketas, kuriame laikomos bibliotekų generavimo klasės

        - **TypeDefinitors** – tipų aprašų generavimo klasės skirtos UML modelio elementų generavimui.
        - **MemberImplementors** – UML elementų sudedamųjų komponenčių generavimo klasės.
- **AssemblyGenerator** – šiame pakete realizuotos klasės .NET dinaminių bibliotekų generavimui.
    - **TypeDefinitors** – pakete laikomos klasės tipų, tokių kaip klasė, interfeisas, numeracija, generavimui.
    - **MemberImplementors** – pakete laikomi tipų sudėtinių dalių, tokių kaip metodas, laukas, savybė, generavimui.

3.5. Architektūros kokybė

Kuriant sistemą daug dėmesio buvo skiriama plečiamumui ir pakartotiniam panaudojimui. Projektuojant architektūra buvo stengiamasi pasiekti kuo aukštesni abstrakcijos lygį. Pagal prasmę ir panaudojimo būdą sistemos elementai išskaidyti į atskiras klasių bibliotekas (angl. class library). Komponentai tarpusavyje bendrauja interfeisų pagalba. Daugumai interfeisų praplėtimo bibliotekoje yra aprašytos bazinės klasės realizuojančios dalį funkcionalumo.

Įrankis turi parametrizuotą klasių bibliotekų generatorių, kurio pagalba buvo sugeneruota UML metamodelio klasių biblioteka. Generavimas užtikrina mažesnę žmogiškos faktoriaus klaidos tikimybę ir garantuoja, kad visos klasės bus aprašytos pagal tą pačią tvarką. UML modelis į sistemą integruojamas naudojant refleksiją, todėl pergeneravus modelį be didesnių sunkumų senąją versiją galima pakeisti naujesne.

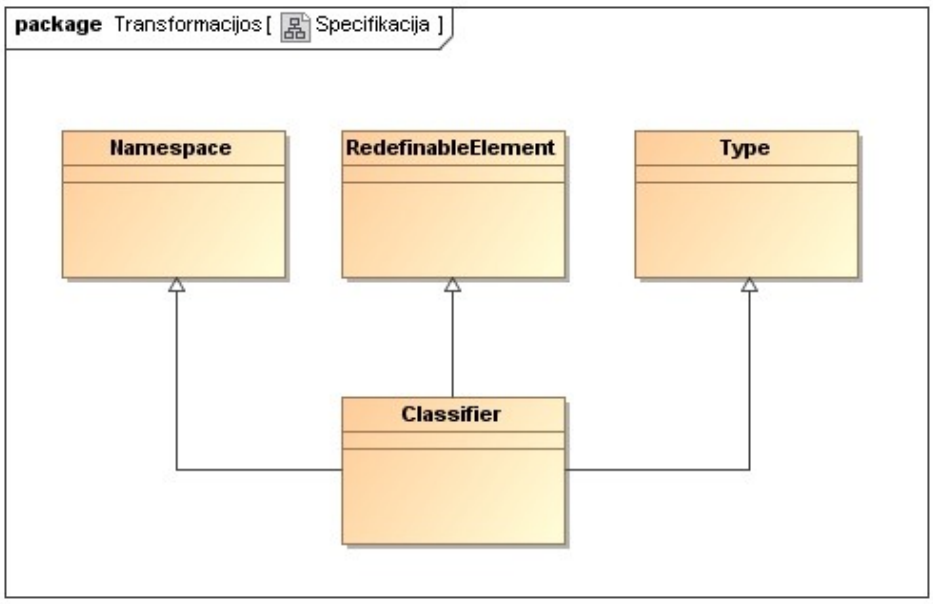
4. Tyrimo dalis

Kuriant UML redaktorių buvo susidurta su modelio elementų realizavimo sunkumais. UML 2 standarto specifikacijos numato per 200 skirtingų elementų tipų. Elementai išskaidyti į daugybę paketų, kurie realizacijoje turi būti suliejami. Specifikacijose daugelis elementų paveldi iš daugiau nei vienos tėvinės klasės. Projekto realizacijai pasirinktas .NET karkasas leidžia klasei paveldėti tik iš vienos tėvinės klasės. Taigi iškilo dvi problemos:

- tiesiogiai UML specifikacijas atitinkančią metamodelio realizaciją riboja .NET karkaso specifika;
- UML metamodelis yra per didelis ir per daug painus, kad būtų įmanoma sukoduoti vienam programuotojui per projektui skirtą laiką.

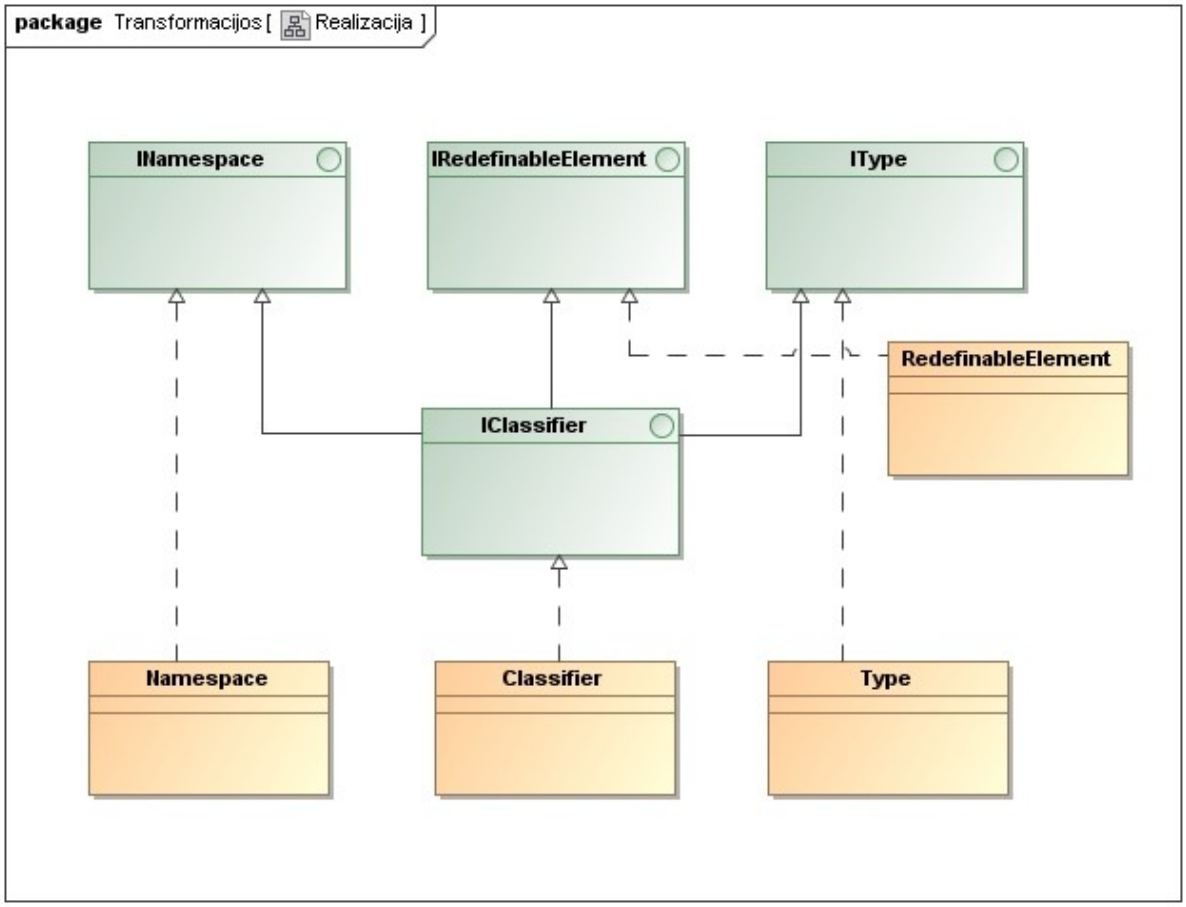
Pirmoji problema išspręsta interfeisų pagalba. .NET karkasas interfeisams leidžia daugybinį paveldėjimą. Sprendimo metodą sudaro trys etapai:

- kiekvienai metamodelio klasei sukuriamas interfeisas aprašantis jos atributus ir metodus;
- remiantis specifikacija interfeisai susiejami paveldėjimo ryšiais;
- kiekviena klasė paveldi jai atitinkantį interfeisą ir realizuojami jos, bei interfeisų pagalba paveldėti atributai ir operacijos.



Paveikslas 8. UML specifikacijose pateikiamo daugybinio paveldėjimo pavyzdys

Atlikus aprašytas transformacijas gaunamas toks vaizdas:



Paveikslas 9. UML modelio elementų vaizdas atlikus transformacijas

Kad išspręsti UML metamodelio realizavimo apimties didumo ir sudėtingumo problemą, elementus buvo nutarta generuoti. OMG grupė, kartu su UML 2.1 specifikacijomis, pateikia ir XMI 2.1 standarto bylas, aprašančias UML metamodelio struktūrą skirtingais meta lygmenimis. OMG grupė, UML specifikacijose, standartą aprašo pačia UML kalba. UML 2.1 standarto specifikaciją sudaro du pagrindiniai dokumentai – infrastruktūros dokumentas ir pilno modelio (angl. Superstructure) dokumentas. Infrastruktūroje aprašomi baziniai UML elementai, o pilno modelio dokumente, remiantis infrastruktūros modeliu, specifikuotas pilnas UML modelis. Taigi, žiūrint iš pilno modelio pusės, infrastruktūros specifikacija aprašo pilno modelio metamodelį.

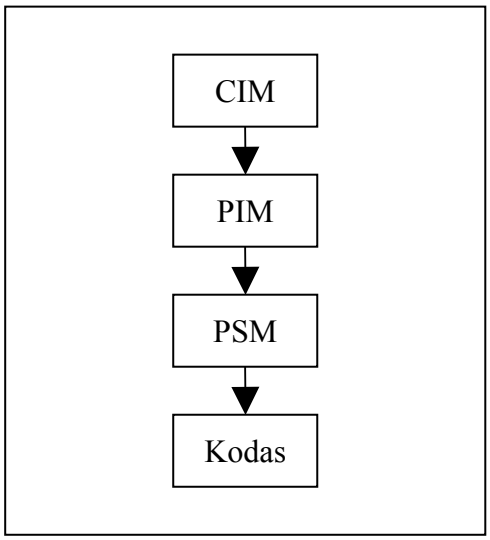
Kodo generavimui iš modelio OMG grupė yra sukūrusi modeliu paremtos architektūros metodą MDA [7] (angl. Model Driven Architecture). Šis metodas aprašo kaip, pasitelkus transformacijas, iš modelio gauti programinį kodą. MDA metodas, neskaitant kodo, numato tris modelius: CIM, PIM, PSM ir tris transformacijas: CIM → PIM, PIM → PSM ir PSM → kodas.

MDA modeliai:

- CIM – nuo skaičiavimų nepriklausomas modelis;
- PIM nuo platformos nepriklausomas modelis;
- PSM – nuo platformos priklausantis modelis.

CIM aprašo organizacijos veiklos modelį. Šio modelio transformavimas į PIM modelį kol kas atliekamas rankiniu būdu. Transformacijos PIM → PSM ir PSM → Kodas atliekamos UML profilių ir stereotipų pagalba.

MDA paradigma sparčiai populiarėja tarp CASE įrankių gamintojų. OMG grupės puslapyje yra užregistruota daugiau nei 50 įrankių vienaip ar kitaip palaikančių MDA metodą [7], tačiau „CASE įrankių, kurie palaikytų pilną MDA transformacijų rinkinį, deja nėra“ [7].



Paveikslas 10. MDA modelių transformacijos

Išanalizavus UML metamodelio generavimo galimybes priimtas sprendimas pasirašyti savo generatorių. Šis tikslas realizuotas keturiais etapas:

1. Sukoduotas infrastruktūroje aprašomą UML metamodelis.
2. Sukoduotas XMI 2.1 standarto bylų nuskaitymo komponentas.
3. Nuskaityta pilno modelio XMI byla.
4. Sugeneruota dinaminė biblioteka su pilno modelio elementais.

Sugeneruotas UML metamodelis sėkmingai integruotas į kuriamą modeliavimo sistemą. Generavimas ženkliai palengvino realizavimo darbus. Redaktoriaus karkase realizuotas funkcionalumas konfigūracijos pagalba leidžia nurodyti kokių elementų savybes leisti redaguoti vartotojui, tačiau sugeneruotas modelis neturi jokių metodų šiom savybėm priskiriamų reikšmių teisingumui ir logiškumui patikrinti. Dėl elementų gausos validavimo metodų kodavimas kiekvienam jų gali užtrukti labai ilgai. Todėl šiam tikslui bus atliekamas tyrimas OCL apribojimų taikymui modelio elementų validavimui.

4.1. OCL išraiškomis paremtas metamodelio objektų validavimas

UML modelio elementų savybių reikšmes riboja tik jų tipai. Pavyzdžiui kiekvienas UML elementas turi savybę „owner“ kuriai kaip reikšmę galima priskirti bet kurį modelio elementą. Leidžiant vartotojui elementų savybėms priskirti pageidaujamas reikšmes galima sulaukti nepageidaujamų rezultatų. Pavyzdžiui vartotojas gali nurodyti, kad elemento savininkas yra pats elementas arba jam priklausantis elementas. Bandant išsaugoti tokį modelį dėl rekursijos įvyks steko perpildymas ir programa turės baigti darbą arba vykdomą operaciją klaidos pranešimu.

Egzistuoja komercinių įrankių, leidžiančių aprašyti OCL apribojimus modelio elementams. Šiuo tyrimu taikysime OCL apribojimus metamodelio elementų savybėms tikrinti.

OCL pagal apibrėžimą yra tipizuota apribojimų kalba. Ja galima užrašyti logines išraiškas, kurios susiejamos su kontekstu: klasifikatoriumi, savybe ar operacija. Tikrinant šių išraiškų teisingumą galima sužinoti ar elementas tenkina jam keliamus reikalavimus.

OCL išraiškų tipai skirstomi į 4 kategorijas:

- Invariantai – šios išraiškos turi būti tikrinamos ir privalo būti teisingos visą objekto gyvavimo ciklą.
- „Pre“ išraiškos – tikrinamos prieš vykdant metodą.
- „Post“ išraiškos – tikrinamos po metodo įvykdymo.
- „Body“ išraiškos – skirtos aprašyti metodo vykdomom operacijom ir yra naudojamos užklausom rašyti.

Modelio elementų savybių reikšmių priskyrimas ir gavimas vyksta per pagalbinius metodus, todėl validavimui galima pritaikyti „pre“ ir „post“ išraiškas. Infrastruktūros ir pilno modelio XMI bylose, kartu su elementais, pateikiami ir jų apribojimai, tačiau, neatlikus pakeitimų, jie gali būti panaudoti tik bendrai elemento ar modelio validacijai.

5. Eksperimentinė dalis

5.1. Automatinė OCL išraiškomis paremta validavimo sistema

Validavimo sistemai realizuoti pirmiausia reikėjo turėti OCL parserį. Parserio generavimui buvo panaudotas „GOLD Parser Builder“ įrankis ir „Calitha“ variklis parserio kodo generavimui C# kalba. OCL gramatikos šiam parserio generatoriui rasti nepavyko, tad eksperimentui buvo realizuotas siauras OCL kalbos gramatikos taisyklių poaibis. Gramatika pateikiama priede.

Buvo realizuotos taisyklės leidžiančios:

- Palyginti ar dviejų argumentų reikšmės yra lygios
- Palyginti ar dviejų argumentų reikšmės nėra lygios
- Palyginti pirmojo argumentų reikšmė didesnė už antrojo
- Palyginti pirmojo argumentų reikšmė didesnė arba lygi antrojo reikšmei
- Palyginti pirmojo argumentų reikšmė mažesnė už antrojo
- Palyginti pirmojo argumentų reikšmė mažesnė arba lygi antrojo reikšmei

Realizuotas gramatika leidžia kaip argumentą perduoti šias reikšmes:

- Sveikas skaičius
- Slankaus kablelio skaičius
- Teksto eilutė
- Loginė boolean reikšmė
- Išraiškos tikrinimo rezultatas
- Elemento atributas
- Elemento metodas

Aprašyta gramatika leidžia išraiškas apjungti AND, OR, XOR bei implikacijos jungtimis, leidžia grupuoti skliaustų pagalba.

5.1.1. Realizavimo ypatumai

Realizuota sistema testuojamus elementus atsirenka refleksijos pagalba. Sistema vykdomos programos moduliuose ieško klasių pažymėtų „ValidationEntity“ žyme.

```
[ValidationEntity]
public class Studentas
```

Paveikslas 11. Validuojamų elementų tipų žymėjimas

Validuojamų objektų savybių reikšmių priskyrimo metodai, prieš priskirdami naują reikšmę, patikrina ar ji nėra lygi senajai ir, kreipdamiesi į elemente statiškai aprašytą validavimo metodą, nustato ar reikšmė yra validi. Jeigu abi šios sąlygos tenkinamos reikšme priskiriama. Validavimo metodai realizuoti statiškai tam, kad vienu validavimo metodo priskyrimu būtu galima tikrinti visus šio elemento tipo objektus.

```
string FirstName
{
    get { return FirstNameField; }
    set {
            if (FirstNameField != value &&
                   OnValidatePropertyValue(this,
                     new ValidatePropertyValueEvent("FirstName",
value)))
                {
                    FirstNameField = value;
                }
        }
}
```

Paveikslas 12. Elementų savybių reikšmių validavimo iškvietimas

5.1.2. Bandymai

Sukompiliuota testuojamų objektų klasių biblioteka prie testavimo sistemos prijungiama dinamiškai. Refleksijos pagalba atrenkami validuojamų objektų tipai ir prijungiami validavimo metodai. Sukūrus validuojamų tipų objektus ir keičiant jų savybes tikrinimas perduodamas validavimo sistemai, ir, jeigu visos OCL taisyklės tenkinamos, savybės reikšmė pakeičiama.

5.1.3. Eksperimento išvados

Validavimas OCL pagalba labai patogus. OCL išraiškos intuityviai suprantamos, be to, validavimo metodų nereikia kompiliuoti kartu su sistema. Validavimo išraiškas galima redaguoti arba prijungti naujas tiesiogiai prie vykdomos sistemos.

Realizavus validavimo sistemą prieita išvados, kad patį validavimą galima atlikti bet kokiems, net ir ne pagal iš anksto numatytą architektūrą aprašytiems tipams. .NET refleksijos pagalba galima perimti elementų savybių priskyrimo ir gavimo metodus ir dinamiškai įterpti validavimą.

5.2. Kompiliuotų ir generuotų elementų spartos palyginimas

Projekto realizavimo metu buvo padaryta prielaida, kad generuotų dinaminių bibliotekų elementai veikia sparčiau negu kompiliuotų. Spartos palyginimui pasirinkta elementų savybių (angl. property) reikšmių gavimas ir priskyrimas.

Testavimui paruoštos dvi bibliotekos su analogiškais elementais. Viena jų sugeneruota sukurtu įrankiu, kita – kompiliuota standartiniu C# kompiliatoriumi be derinimo žymių. Žvelgiant į bylų dydi pastebimas skirtumas: generuota – 5.632, kompiliuota 20.480 baitų. Dekompiliavus bibliotekas atkreiptas dėmėsis į tai, kad elementų savybių reikšmių priskyrimo ir gavimo metodus sudaro

5.2.1. Testai

1. **Integer Set** – sveiko skaičiaus priskyrimo savybei testas. Priskiriama ciklo skaitiklio reikšmė.
2. **Integer Get** – sveiko skaičiaus gavimo iš savybės testas.
3. **String Set** – teksto eilutės priskyrimo savybei testas. Reikšmė atrenkama iš dviejų galimų imat ciklo skaitiklio modulį iš 2.
4. **String Get** – teksto eilutės gavimo iš savybės testas.
5. **Boolean Set** – boolean tipo reikšmės priskyrimo savybei testas. Reikšmė atrenkama iš dviejų galimų atrenkant pagal ciklo skaitiklio modulį iš 2.
6. **Boolean Get** – boolean tipo reikšmės gavimo iš savybės testas.

7. **Reference Set** – nuorodos į objektą priskyrimo savybei testas. Reikšmė atrenkama iš dviejų galimų imat ciklo skaitiklio modulį iš 2.
8. **Reference Get** – nuorodos į objektą gavimo iš savybės testas.

5.2.2. Bandymų aplinkos

Visuose testavimo aplinkose buvo naudojamos Microsoft .NET 2.0.50727 karkaso ir Microsoft Visual C# 2005 8.00.50727.3053 kompiliatoriaus versijos.

Testavimo sistema 1

Centrinis procesorius:	AMD Athlon 2800+, 2.08GHz
Operatyvinė atmintis:	1.75GB
Operacinė sistema:	Microsoft Windows XP SP3
C# kompiliatoriaus versija:	Microsoft Visual C# 2005 8.00.50727.3053

Testavimo sistema 2

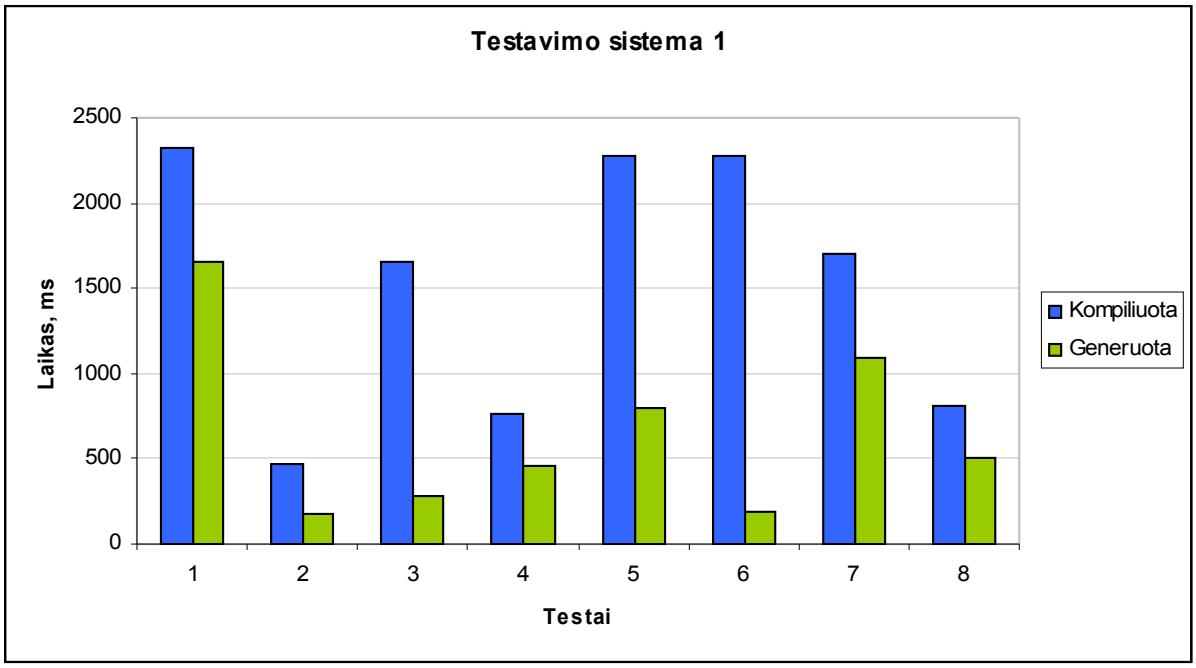
Centrinis procesorius:	Intel Core 2 Quad Q6600, 2.40GHz
Operatyvinė atmintis:	3GB
Operacinė sistema:	Microsoft Windows XP SP2
C# kompiliatoriaus versija:	Microsoft Visual C# 2005 8.00.50727.1433

5.2.3. Bandymų rezultatai

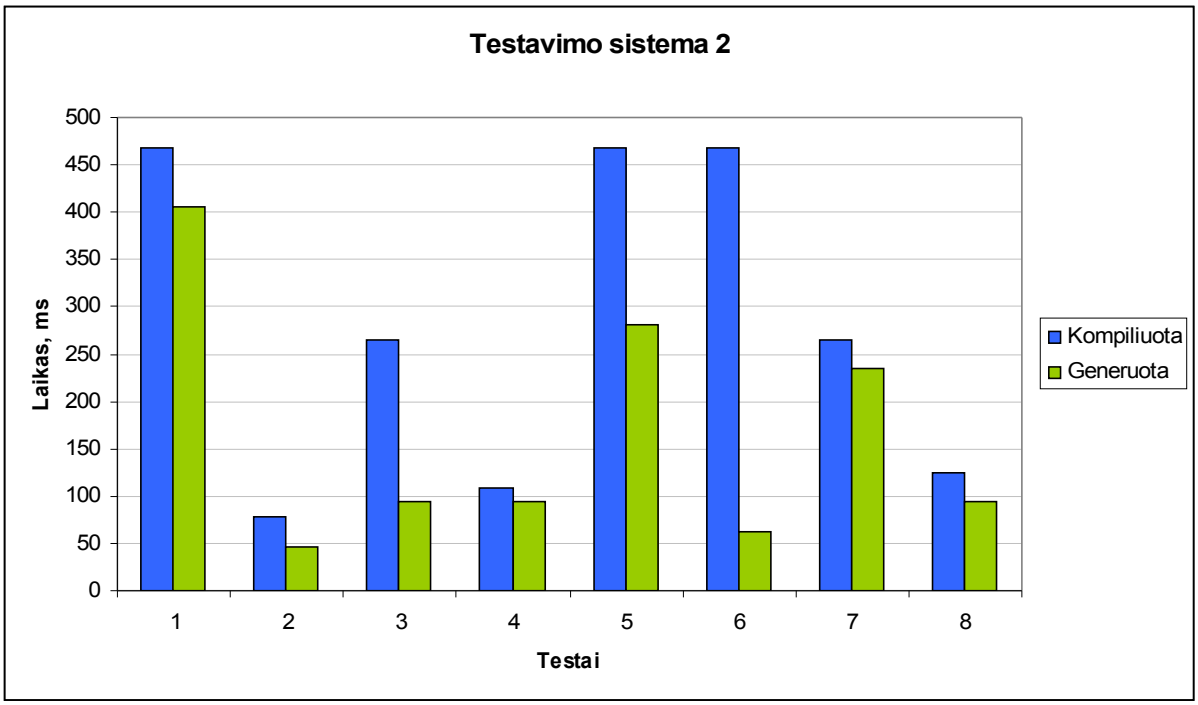
Kiekvienas testas abejose testavimo sistemose buvo atliktas 10000000 kartų ir vykdymui užtrūkęs laikas užrašytas lentelėje milisekundėm.

Lentelė 2. Generuotų ir kompiliuotų dinaminių bibliotekų spartos testų rezultatai

Sistemos / Testai		Testavimo sistema 1		Testavimo sistema 2	
		Kompiliuota	Generuota	Kompiliuota	Generuota
Integer	Set	2328,125	1656,25	468,75	406,25
	Get	468,75	171,875	78,125	46,875
Text	Set	1656,25	281,25	265,625	93,75
	Get	765,625	453,125	109,375	93,75
Boolean	Set	2281,25	796,875	468,75	281,25
	Get	2281,25	187,5	468,75	62,5
Reference	Set	1703,125	1093,75	265,625	234,375
	Get	812,5	500	125	93,75



Paveikslas 13. Pirmosios testavimo sistemos rezultatai



Paveikslas 14. Antrosios testavimo sistemos rezultatai

5.2.4. Bandymų išvados

Susumavus kompiliuotos ir generuotos bibliotekų testų laikus gauta, kad generuotos bibliotekos testų vykdymas užtruko apytiksliai du kartus trumpiau.

Testavimo metodai kiekvienos iteracijos metu turėjo atlikti papildomus veiksmus reikšmių gavimui ar priskyrimui, todėl, sumažinus pašalinių veiksnių įtaką testams, realu tikėtis dar geresnių rezultatų generuotos bibliotekos naudai.

6. Išvados

Projekto vykdymo eigoje sukurtas modeliavimo karkasas realizuojantis pagrindines modelių bei diagramų redagavimo funkcijas ir palengvinantis tolesnį redaktoriaus kūrimą. Sukurtas redaktorius ir karkasas, kaip pagrindas, naudojamas kito magistrinio darbo projektinės dalies, SCRALL redaktoriaus, realizacijai.

Sukurtas tiesioginis UML modelių struktūros transformavimo į .NET klasių bibliotekas metodas. Ištyrus šiuo būtu sugeneruotas ir standartiniu būdu kompiliuotas klasių bibliotekas nustatyta, kad generuotų bibliotekų metodai vykdomi apytiksliai dvigubai sparčiau.

Surastas būdas ir sukurta OCL išraiškomis paremta objektų savybių validavimo sistema, leidžianti keisti validavimo taisykles programos vykdymo metu. Ši sistema bus integruota į redagavimo įrankį.

Projekto realizavimo metu didelės dalies kodavimo darbų buvo išvengta pasinaudojus sistemos teikiamomis tiesioginės inžinerijos galimybėmis. Tai patvirtina įrankio naudingumą

7. Literatūra

1. OMG grupė. OMG Unified Modeling Language Infrastructure Version 2.2. *Standarto specifikacija*. 2008
2. OMG grupė. OMG Unified Modeling Language Superstructure Version 2.2. *Standarto specifikacija*. 2008
3. OMG grupė. UML 2.0 OCL Specification. Standarto specifikacija. 2007
4. Pavlov, V. „The Babel Experiment“: An Advanced Pantomime-based Training in OOA&OOD with UML: ACM Technical Symposium on Computer Science Education. Misūrio valstija JAV, 2005.
5. Meyer, B. UML: The Positive Spin. *American Programmer*, 1997. Prieiga per internetą: http://archive.eiffel.com/doc/manuals/technology/bmarticles/uml/page.html
6. OMG grupė. MDA. Prieiga per internetą: http://www.omg.org/mda/
7. Tutkutė, L. MDA ir programų kodo generavimas. *Referatas*. KTU Informatikos fakultetas, 2008.
8. OMG, Commited Companies http://www.omg.org/mda/committed-products.htm
9. GOLD Parsing System. Prieiga per internetą: http://www.devincook.com/goldparser/

8. Terminų ir santrumpų žodynas

OMG (angl. Object Management Group) – konsorciumas, kurio pagrindinis tikslas standartų rengimas objektiškai orientuotų paskirstytųjų sistemų.
UML (angl. Unified Modeling Language) – vieninga modeliavimo kalba.
XML (angl. eXtensible Markup Language) – bendros paskirties duomenų struktūrų bei jų turinio aprašomoji kalba.
XMI (angl. XML Metadata Interchange) – XML meta duomenų tarpusavio apsikeitimas.
MOF (angl. Meta-Object Facility) – standartas parengtas OMG konsorciumo.
OCL (angl. Object Constraint Language) – objektų apribojimo kalba.
CIL (angl. Common Intermediate Language) – bendra tarpinė kalba, >NET karkaso pagrindas
MDA (angl. Model Driven Architecture) – modeliu paremta architektūra tai metodas aprašantis modelių transformacijas į kodą.
CIM (angl. Computation Independent Model) – nuo skaičiavimų nepriklausomas modelis.
PIM (angl. Platform Independent Model) – nuo platformos nepriklausomas modelis.
PSM (angl. Platform Specific Model) – nuo platformos priklausomas modelis.
Klasių biblioteka (angl. Class Library) – sąvoka vartojama kalbant apie .NET dinamines bibliotekas.
Delegatas (angl. Delegate) – savybė, kurios reikšmė tam tikro tipo metodų masyvas. Naudojama įvykiams iššaukti

9. Priedai

9.1. Įrankyje naudota OCL paremta gramatika

```
"Name"     = 'OCL'
"Author"   = 'Vytautas Barkauskas'
"Version"  = '0.1'
"About"    = 'Simple OCL-based grammar'

"Start Symbol" = <Expression>

! ------------------------------------------------ Sets

{ID Head}      = {Letter} + [_]
{ID Tail}      = {Alphanumeric} + [_]
{String Chars} = {Printable} + {HT} - ["\]
{PlusMinus}    = [+-]
! ------------------------------------------------ Terminals

Identifier        = {ID Head}{ID Tail}*
!MemberName       =  '.' {ID Head} {ID Tail}*
StringLiteral     = '"' ( {String Chars} | '\' {Printable} )* '"'
IntegerLiteral   = {Number}+ | {PlusMinus} {Number}+
RealLiteral       = {Digit}*'.'{Digit}+ | {PlusMinus} {Digit}*'.'{Digit}+
Null              = 'null'
BooleanLiteral = 'true' | 'false'
Equal             = '='
NotEqual          = '<>'
MoreThan          = '>'
MoreOrEqualThan   = '>='
LessThan          = '<'
LessOrEqualThan   = '<='

And = 'and'
Or = 'or'
Xor = 'xor'
Implies = 'implies'
!Not = 'not'

Self = 'self'

OCLCall = '->'

! ------------------------------------------------ Rules



<Expression>              ::= <Value> <Operator> <Value>
                            | <Combined Expression>
                            | <Priority Expression>

<Combined Expression> ::= <Expression> <Connector> <Next Expression>
<Priority Expression> ::= '(' <Expression> ')'
```

```
<Next Expression> ::= <Expression>

<Connector>                ::= And
                           | Or
                           | Xor !Atskirti
                           | Implies


<Value>                   ::= IntegerLiteral
                           | StringLiteral
                           | BooleanLiteral
                           | RealLiteral
                           | <Member Path>
                           | <OCL Operation>
                           | Null

<Operator>                 ::= Equal
                           | NotEqual
                           | MoreThan
                           | MoreOrEqualThan
                           | LessThan
                           | LessOrEqualThan
                           | Implies !Atskirti




<Member Path> ::= <Member>
                  | Self '.' <Member>
                  | <Member Path> '.' <Member>

<Member> ::= <Attribute>
             | <Operation>

<Attribute> ::= Identifier

<Operation> ::= Identifier '(' <Parameter List> ')'

<OCL Operation>       ::= <Member Path> OCLCall Identifier '(' <Parameter
List> ')'



<Parameter List>           ::= <Value>
                           | <Parameter List> ',' <Value>
                           | !Nothing
```