T A R T U Ü L I K O O L

MATEMAATIKA-INFORMAATIKATEADUSKOND

Arvutiteaduse instituut

Tarkvarasüsteemide õppetool

Natalia Tomassova

Grammatika-põhine konverteriarendus

Väitekiri *magister scientarium* kraadi taotlemiseks informaatika erialal

Juhendaja: prof. Jüri Kiho

TARTU 2005

Sisukord

Sissejuhatus

Proloog

Tänapäeval puututakse tihti kokku erinevate arvutitekstidega (tekstilisel kujul tarkvaraga), mida soovitakse mingil viisil töödelda, kas siis redigeerida, lugeda, teisendada vms. Arvutitekstid võivad omada erinevat süntaktilist struktuuri ning nende töötlemine võib nõuda spetsiifiliste funktsioonide olemasolu. Eritoimetite "algusest peale" loomine või olemasolevate "valmis" süsteemide kohandamine/integreerimine on enamasti kulukas. Üheks alternatiiviks on skeemtehnoloogia ja sellel baseeruv skeemtoimeti *Amadeus* [1].

Käesolev töö keskendub arvutitekstide teisendamise teatavale eriliigile – konverteerimisele. Konverteerimise all mõistetakse arvutitekstide "kergekaalulist" transleerimist, mille juures põhirõhk asetub just tekstide vormi teisendamisele. Näitena võib nimetada mingi programmeerimiskeele lähtekoodi viimist *XML* kujule.

Üha suurenev vajadus erinevate tüüpi konverterite järele on tõstatanud loomuliku küsimuse, kuidas automatiseerida konverterite loomist. Muidugi on soovitav luua võimalikult üldist tüüpi konvertereid.

Süsteemile *Amadeus* on konverteerimine seesmiselt omane. Toimetatavaid tekste käsitletakse alati spetsiaalsel sisekujul, nn. skeemkujul ehk skeemtekstidena. Sidepidamiseks "lihttekstide maailmaga" peavad aga süsteemis leiduma võimalused lihttekstide teisendamiseks (konverteerimiseks) skeemtekstideks ja vastupidi. See asjaolu teebki võimalikuks vaadelda süsteemi *Amadeus* sobiva platvormina mitmesuguste eriotstarbeliste konverterite arendamiseks.

Selleks, et süsteem *Amadeus* töötaks konverterina mingist lähtekeelest

LK süsteemi skeemkeelde SK, peab olema süsteemi integreeritud lähtekeelele vastav nn. baaskeel. Integreerimine tähendab muuhulgas nii skeemistuskonverteri (ehk skeemisti) $LK \rightarrow SK$ kui ka vastupidise, $SK \rightarrow LK$ konverteri, nn. tekstualiseerija realiseerimist. Kui viimane on suhteliselt kergesti teostatav, siis skeemistuskonverteri "käsitsi" loomine on ülimalt töömahukas protsess. Selle juures on paratamatult vaja lähtuda konverteeritava keele LK formaalsest grammatikast.

Käesolev väitekiri

Käesolevas töös keskendutaksegi just skeemistuskonverterite loomise automatiseerimisele – küsimusele, kuidas uue baskeele LK korral võimalikult efektiivselt välja töötada vastav skeemistamise meetod, mis realiseeriks konverteri tüüpi $LK \rightarrow SK$ (lähtekeel $\rightarrow$ skeemkeel). Töö tulemusena loodud, vastavate automatiseerimisvahenditega täiendatud süsteemi *Amadeus* uus versioon nimetusega *AmadeusJSN* [3] kujutab endast juba küllaltki efektiivset konverteriarenduse tarkvara. Automatiseeritud on uue baaskeele lisamine (ka baaskeele kustutamine ning taastamine). Skeemistuskonverteri loomiseks tuleb kasutajal vaid vajalikul kujul kirjeldada lähtekeele grammatika. Tulemusena saab süsteemi kasutada $LK \rightarrow SK$ tüüpi konverterina, ja seda hõlpsasti varieeritavate teisenduseeskirjade (skeemkonversiooni mudelite) alusel.

Märgime, et üldist tüüpi $LK \rightarrow TK$ konverteri realiseerimiseks tuleb süsteemi integreerida nii lähtekeel LK kui ka tulemuskeel TK, kusjuures viimase jaoks pole skeemistuskonverterit vaja luuagi.

Käesolev väitekiri koosneb neljast peatükist.

Esimeses peatükis tutvustatakse erinevaid olemasolevaid konverteerimise süsteeme: selliseid nagu *DEL*, *LinkGrammar*, *XML Convert*. Selgitatakse tarkvara konverteerimise mõistet. Püstitatakse ja konkretiseeritakse konverterite loomise automatiseerimise probleem ning esitatakse selle lahendamise üldine idee.

Väitekirja teises peatükis esitatakse süsteemi *Amadeus* määratlus ja tutvustatakse selle põhifunktsioone. Antakse ülevaade skeemteksti mudelist ja skeemtöötluse eesmärgist.

Kolmandas peatükis kirjeldatakse üldist põhimõtet, kuidas süsteemis *Amadeus* mingi antud lähtekeele *LK* jaoks luuakse skeemisti, st. konverter keelest *LK* skeemkeelde *SK*, lähtudes keele *LK* grammatikast. Lihtsa näitekeele – metsa vasaku suluesituse keele (*VSE*) – najal kirjeldatakse uue baaskeele lisamise protsessi, eeskätt spetsifitseeritakse kaasnevad tegevused ning nende järjekord.

Väitekirja neljandas peatükis esitatakse ülevaade peamistest süsteemile *AmadeusJSN* omastest, uude versiooni lisandunud tehnilistest lahendustest. Selgitatakse, kuidas on automatiseeritud baaskeeltega manipuleerimine (uue baaskeele lisamine, olemasoleva kustutamine, kustutatud baaskeele taastamine). Grammatikafaili iteratiivse arendamise toeks spetsifitseeritakse vastav projektikirjeldus. Peatüki lõpus tutvustatakse lähemalt uue süsteemiversiooni testimist näitekeelte *VSE* ja *PSE* (metsa parem suluesitus) baasil.

Lisades paiknevad mitmesugused skeemkujul koodiosad ning muud tehnilisemat laadi andmed. Viimases lisas (CD andmekandjal) leidub kogu uus süsteem *AmadeusJSN* koos käesoleva magistritöö e-koopiaga.

Väitekiri on kirjutatud ja vormistatud süsteemide *Amadeus* ja LaTeX abil.

Peatükk 1

Konverteerimine ja konverteriarendus

Käesolevas peatükis selgitatakse tarkvara konverteerimise mõistet. Tutvustatakse mõningaid olemasolevaid konverteerimise süsteeme. Püstitatakse ja konkretiseeritakse konverterite loomise automatiseerimise probleem ning esitatakse selle lahendamise üldine idee.

1.1 Konverteerimise mõiste määratlus

Kõige üldisemas mõttes tähendab tarkvara konverteerimine mingitele tarkvaralistele lähteobjektidele uue vormi andmist. Antud töös piirdume tekstiliste objektidega, nn. arvutitekstidega [1]. Seega konverteeritavateks olemiteks on tekstid mingis lähtekeeles (LK) ja tulemusteks tekstid mingis sihtkeeles (SK). Konkreetne, teatud liiki konverteerimine on funktsioon $LK \rightarrow SK$. Konverteerimine (kui eeskätt vormi muundamine) on transleerimine kitsamas tähenduses: teisendatakse ainult teatavaid kõrgema taseme struktuure (programmeerimise keele puhul nt. juhtimiskonstruktsioonide kuju), jättes primitiivsemad konstruktsioonid (nt. avaldised) muutmata.

Konverteerimise liigi ehk tüübi määrab vastav reeglistik. Lähtekeele teksti (teatavat liiki) konverteerimine tähendab selle teisendamist teise vormi vastavalt (teatavale) konverteerimisreeglite hulgale. Konverteerimisreeglid defineerivad, kuidas ja kus kohas sihttekstis paigutuvad lähteprimitiivid. On mõist-

lik vaadelda iga konverteerimisreeglite hulka (vastavat liiki) konverteerimise mudelina. Seega konverteerimise meetod on määratud lähtekeele, sihtkeele ja konverteerimise mudeliga. Kuna konverteerimine tähendab lihtsalt formaadi/kuju muutmist, siis mõlematel, nii lähte- kui ka sihtkeelel, on ühed ja samad primitiivid. Mõnel juhul defineerib mudel kahepoolse konverteerimise: nii lähtekeelest sihtkeelde kui ka vastupidi.

Käesoleval ajal on üheks populaarsemaks konverteerimise liigiks tekstide teisendamine *XML* kujule (üldnimetus ingl. k. *Some-to-XML*): ühe või teise keele tekste "kodeeritakse" *XML* konstruktsioonidena (näiteks *Java-to-XML* [11], *Flat-to-XML* [9]). Efektiivsemad konverterid töötlevad grammatiliselt keerukamaid lähtekeeli. Mõned lihtsamad konverterid aga seavad lähtekeelele küllaltki suuri kitsendusi; tüüpnäiteks võib tuua nõude, et lähteandmed oleksid kujutatud nn. *CSV* (*comma separated*) väärtuste väljadena. Ilmselt sellised piirangud ei ole vastuvõetavad paljude tavaliste tarkvarastruktuuride korral.

Some-to-XML tüüpi konverteerimine kombinatsioonis *XML-to-Some* tüüpi konverteerimisega annab konverteerimise tüüpi *Some-to-Some*. Tuleb tähele panna, et just *XML* kasutamine vahepealse sihtkeelena mõnedes *Some-to-Some* konverterites ei ole hädavajalik. Alternatiivselt võib *XML* asemel kasutada ka mõnda muud "vahekeelt" või loobuda üldse vahekeelest ning vahetult realiseerida vajalik *Some-to-Some* konverter.

Leidub ka *Any-to-Some* tüüpi konverteerimise süsteeme, mis on võimelised töötama mitme erineva lähtekeelega.

1.2 Konverteerimise süsteemide näiteid

1.2.1 *DEL*

Korporatsioonis *Republica Corp.* on välja töötatud *Any-to-XML* konverteerimise tehnoloogia, mille aluseks on spetsiaalne andmete ekstraheerimise keel *DEL* (*Data Extraction Language*) [10]. Keel *DEL* on *XML*-põhine ning kirjeldab andmete konverteerimise protsessi teistest andmeformaatidest *XML* kujule. *DEL* skript spetsifitseerib, kuidas paigutada ja eraldada sisendandmete fragmente ja kuhu kohta on neid vaja lisada *XML* tulemusfailis. *DEL* protses-

sori *DEL* skriptile rakendamise tulemusel ekstraheeritakse lähtetekstist andmeid, mida võib kasutada nii uue *XML* dokumendi loomiseks kui ka juba olemasoleva *XML* dokumendi modifitseerimiseks. Uued elemendid ja atribuudid lisatakse kohtadesse, mis on määratud *XPath*[1] avaldistega. *DEL* skript koos algandmetega antakse ette *DEL* protsessorile, mis teeb tegeliku andmete konverteerimise vastavalt skriptile. *DEL* protsessori väljundiks on trimmis (*well-formed*) [15] *XML* dokument, mis sisaldab soovitud viisil paigutatud algandmete (lähteteksti) fragmente. Andmefragmentide paigutamine tulemusse võib toimuda ka mallide otsimise ja regulaaravaldiste sobitamise teel. Väljaeraldatud andmefragmendid paiknevad ajutiselt *DEL* protsessori registrites (magasinides), neid saab kas kustutada või võimalusel uuesti kasutada otsimismallina. Üldiselt loetakse aga andmed registritest ja paigutatakse vastavatesse asukohtadesse *XML* tulemusdokumendi *DOM*[2] puusse. *XML* dokumendi andmete paigutamise kirjeldamist hõlbustab kursori funktsioon (jooksva positsiooni lokaliseerimiseks).

Järgnevas esitame *DEL* skripti kasutamise näite. Olgu antud reeglistik (punkt 1 alljärgnevas) ekstraheerimaks andmeid tavalisest *HTML* dokumendist (2). Sel korral saadakse tulemuseks *XML* fail (3).

1. *DEL* skript sisaldab järgmisi elemente:

   - *DEL* skripti juurelemendiks on element nimega *wrapper*.
   - Järgmised *map* elemendid loovad juurelemendi nimega *root* ja alamelemendi *description*. Kolmas element *map* lisab veel alamelemendi *Extracted data*.

     ```
     <map maptype="createDocument" node="root"/>
     <map maptype="createElement" node="description"/>
     <map maptype="createTextNode" node="Extracted data"/>
     ```

   - Järgmine element *extract* eraldab sisendandmetest andmeid alates elemendist *<table>*

[1] *XPath* põhiline eesmärk on *XML* dokumendi osade adresseerimine [10].

[2] *Document Object Model* – programmeerimise rakendusliides valiidsete *HTML* ja trimmis *XML* dokumentide jaoks, mis defineerib dokumendi loogilise struktuuri ning dokumendi manipuleerimise ja ligipääsu võtted [10].

```
<extract exptype="over" expression="&lt; table &gt;"/>
```
kuni elemendini *</table>*.
```
<extract exptype=upto" expression="&lt;/table &gt;">
```

- Element *repeat* kordab *DEL* skripti sisemisi elemente nii palju kordi, kui on võimalik.
```
<repeat times="*">
```

- Järgmisena element *extract* eraldab sisendandmetest andmeid esialgu üle *<tr>*, siis üle *<td>* kuni "*<*". Kolmas element salvestab eraldatud andmeid magasini.
```
<extract exptype="over" expression="&lt; tr &gt;"/>
<extract exptype="over" expression="&lt; td &gt;"/>
<extract exptype="upto" expression="&lt;" save="stack"/>
```

- Pärast seda esimene *map* element liigub kursori juurelemendile (*/root*) ja loob elemendi *row* ja *field1* ning sisu paigutatakse magasini.
```
<map maptype="moveCursor" node="/root"/>
<map maptype="createElement" node="row"/>
<map maptype="createElement" node="field1"
content="stack"/>
```

- *Map* element nihutab jooksva kursori positsiooni ühe taseme võrra ülespoole.
```
<map maptype="moveCursor" node=".."/>
```

- Esimene element *extract* eraldab sisendandmetest andmeid, alates *"<td>"* kuni "*<*". Teine element salvestab eraldatuid andmeid registrisse *reg1*.
```
<extract exptype="over" expression="&lt;td&gt;"/>
<extract exptype="upto" expression="&lt;" save="reg1"/>
```

- Järgmine *map* element loob elemendi *field2* ja paneb registri 1 sisu sinna
```
<map maptype="createElement" node="field2"
content="reg1"/>.
```

- *Test* element kontrollib väärtuse "4" võrdsust registri 1 väärtustega. Kui väärtused on võrdsed, siis *map* element loob uue atribuudi nimega *test* ja väärtusega "*success*".

```
<test testtype="equal" value1="4" value2="reg1">
<map maptype="createAttribute" node="test"
content="success"/>
</test>
</repeat>
</extract>
</wrapper>
```

2. Lihtsa *HTML* dokumendi näide:

```
<html>
<body>
<p>Test Material</p>
<table>
<tr><td>Some numbers</td><td>Other numbers</td></tr>
<tr><td>1</td><td>2</td></tr>
<tr><td>3</td><td>4</td></tr>
</table>
</body>
</html>
```

3. *XML* väljund 2. punkti *HTML* dokumendi korral vastavalt 1. punkti *DEL* skripti reeglitele:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<root>
<description>Extracted data</description>
<row> <field1>Some numbers</field1>
<field2>Other numbers</field2>
</row>
<row>
<field1>1</field1>
<field2>2</field2>
</row>
```

```
<row>
<field1>3</field1>
<field2 test="success">4</field2>
</row>
</root>
```

Kokkuvõttes, keel *DEL* on sisuliselt konverterite programmeerimise (küllaltki keeruline) keel. Iga uue konkreetse lähtekeele/mudeli korral tuleb koostada vastav konverteerimise eeskiri *DEL* skripti kujul.

1.2.2 *XFlat* ja *XML Convert*

XML vorm on muutumas prevaleerivaks mitmesuguste rakenduste andmete saatmisel brauseritesse ja ärirakendustesse. Rõhutatakse (nt. [9]), et *XML* on selleks väga sobiv, sest *XML* dokumendid on isekirjelduvad (*self-describing*); nende parsimine[1] on lihtne, samas aga saab sel viisil kirjeldada väga keerulise struktuuriga andmeid. Loomulikult peab *XML*-põhise andmevahetuse (*interchange*) korral saatja-rakendus olema võimeline *XML* dokumendi eksportimiseks ja saaja-rakendus olema võimeline *XML* dokumendi importimiseks. Paljud vanemad rakendused aga kasutavad andmete eksportimiseks ja importimiseks nn. lamefaile (*flat* faile). Nende liidestamisel *XML*-orienteeritud rakendustega tekib vajadus lamefailide konverteerimiseks *XML* dokumentideks, ja vastupidi.

Flat failid

Flat failis (ehk lamefailis) on andmed organiseeritud kindla lihtsa andmeesituse grammatika kohaselt, kusjuures andmed on harilikult kodeeritud üksnes prinditavate märkide (nn. tärkide) abil.

Lamefail koosneb tavaliselt kirjete ehk ridade seeriatest, kus iga kirje on väljade järjend. Väli sisaldab atomaarseid (edasiselt struktureerimata) andmeid. Nii kirjed kui ka väljad eraldatakse üksteisest spetsiaalsete tärkidega – eraldajatega.

Lamefaili näiteks võib tuua isikuandmete *flat* faili, mis sisaldab ühe või mitme isiku andmeid. Iga kirje esitab järgmisi andmeid:

[1] Käesolevas kasutame termini *süntaksanalüüs* asemel ka lühemat sünonüümi *parsimine*.

- isikukood;
- isiku täisnimi jutumärkides (eesnimi on eraldatud perenimest komaga ja sellele järgneva tühikuga);
- isiku palk.

Üks konkreetne isikute lamefail võiks siis välja näha järgmiselt:

```
41212122233, "Carr, Lisa", 10000.00
36012122233, "Barr, Clark", 15000.00
47712122233, "Breaks, Tony", 7100.00
```

Iga kirje sisaldab ühe isiku andmeid. Antud juhul on tegemist nn. *CSV* formaadis andmetega, kuna kirjete eraldajaks on reavahetus ja väljade eraldajaks on koma (kui tühikut mitte arvestada).

Lamefaili struktuur ei pruugi alati olla *CSV*-tüüpi. Keerulisem näide lamefailist: faili struktuur määratletakse sarnaselt *Windows* konfigureerimissätingute failiga (nt. fail *win.ini*). Järgnev näitefail esitab kontaktide nimekirja:

```
[contact]
name=Lisa Carr
email=lisa.carr@hotmail.com
phone=(+372) 555-9328
[contact]
email=barr.clark@yahoo.com
name=Clark Barr
[contact]
phone=(+372) 555-3249
name=Tony Breaks
email=tonyb@bluesuburbanskies.com
```

Iga kontakti kirje algab võtmesõnaga `[contact]` ja sisaldab kolm mittekohustuslikku parameetrit:

- `name =` <isiku nimi>

- `email =` <isiku e-posti aadress>
- `phone =` <isiku telefoninumber>

Kirjete eraldajaks on reavahetus. "Lamedus" antud kontekstis tähendab seda, et fail ei ole indekseeritud ja puudub kirjete hierarhiline struktuur (kirje ei saa paikneda kirjes).

Lamefailide formaadivalik ei piirdu kaugeltki kahe ülaltooduga. Lamefaili soovikohaseid formaate defineeritakse *XFlat* keele abil.

***XFlat* keel**

XFlat on *XML*-põhine keel lamefailide formaati kirjeldavate skeemide (*XML schemas*) defineerimiseks. *XFlat* skeem on *XML* dokument, mis on vastavuses *XFlat* keelega ja kirjeldab *flat* failide klassi formaati. *XFlat* skeemiga saab defineerida sellise lamefaili klassi struktuuri ja süntaksi, mis ise ei sisalda *XML* andmeid. Samuti defineerib *XFlat* skeem *XFlat* isendite klassi struktuuri ja süntaksi. *XFlat* isend on *XML* dokument, mille struktuur on sama, mis *flat* failis ja mille andmed on samad, mis on *flat* faili andmed. *XML* konverter (*XML Convert*), mis on loodud firmas *Unidex*, kasutab *XFlat* skeeme *flat* failide teisendamiseks *XFlat* isenditeks ja vastupidi.

XFlat skeem sisaldab vajalikku informatsiooni vastavas formaadis lamefaili konverteerimiseks *XML* kujule ja vastupidi. *XFlat* on deklaratiivne keel, seetõttu isegi mitteprogrameerija, kes on tuttav *flat* failidega, võib samuti *XFlat* skeeme luua.

Ülaltoodud isikute lamefaili struktuuri kirjeldus *XFlat* skeemina on esitatud joonisel 1.1, kus

`XFlat` on (kohustuslikuks) juurelemediks,

`SequenceDef` on objektide jada näitamiseks,

`RecordDef` on kirjete defineerimiseks,

`FieldDef` on väljade defineerimiseks.

XML Convert

XML Convert on *Java* rakendus, mis kasutab *XFlat*-keelseid skeeme lamefailide konverteerimiseks *XML* dokumentideks (ja vastupidi), teisendades lamefaile ühest formaadist teise. *XML Convert* toetab suurt hulka andme-

```
<?xml version="1.0" ?>
- <XFlat Name="employees_schema" Description="Schema for CSV flat file">
  - <SequenceDef Name="employees" Description="employees flat file">
    - <RecordDef Name="employee" FieldSep="," RecSep="\N" MaxOccur="0">
        <FieldDef Name="ssn" NullAllowed="No" MinFieldLength="9"
          MaxFieldLength="11" DataType="Integer" MinValue="0"
          QuotedValue="Yes" />
        <FieldDef Name="name" NullAllowed="No" QuotedValue="Yes" />
        <FieldDef Name="salary" NullAllowed="No" DataType="Float" MinValue="0"
          QuotedValue="Yes" />
      </RecordDef>
    </SequenceDef>
  </XFlat>
```

Joonis 1.1: Isikute lamefaili struktuuri kirjeldus keeles *XFlat*.

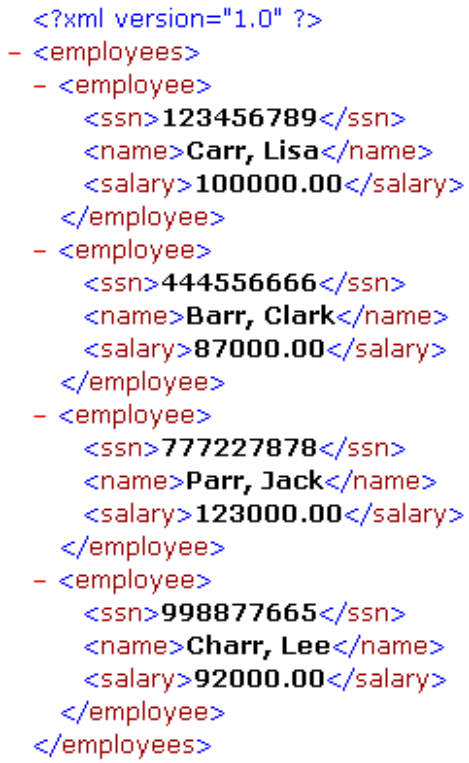
formaate, selliseid nagu *CSV*, fikseeritud pikkusega kirjed ja väljad, kirjete grupp, pesastatud grupid jm. *XML Convert* sooritab ka valiidsuskontrolli: enne tegelikku konverteerimist tehakse kindalaks lähtefaili andmete struktuuri ja väljade tüüpide vastavus *XFlat* skeemile. Kui fail ei läbi sellist kontrolli edukalt, siis konverteerimist ei sooritata (lähtefail lükatakse tagasi). Sellise kontrolliga minimiseeritakse võimalus, et vale struktuuriga/andmetega *XML* dokument saadetakse/imporditakse rakendustesse.

Kasutades *XFlat* skeemi (joonisel 1.1) konverteerib *XML Convert* ülalpool toodud konkreetse isikute lamefaili *XML* dokumendiks, mis on toodud joonisel 1.2.

Süsteemi *XML Convert* rakendatavus piirdub kirje-väli tüüpi struktuuriga lähtetekstidega.

1.2.3 *LinkGrammar*

Link grammatika (*LinkGrammar*, *LG*) [12] on *Carnegie Melloni* ülikoolis välja töötatud formalism, milles lause süntaktiline struktuur esitatakse seoste hulgana (*linkage*). *Link* grammatika koosneb sõnade hulgast (grammatika terminaalsed sümbolid), kus igaühel on olemas seose nõue (*linking requirement*). Sõnade jada on *LG* keele lause siis, kui sõnade vahel on võimalik joonistada seoseid (*links*) nii, et oleksid rahuldatud järgmiseid nõuded:



Joonis 1.2: *XML Convert* töö tulemuse näide.

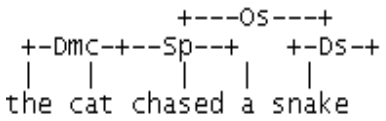
1. Planaarsus. Seosejooned ei lõiku, kui need joonistada sõnade kohale.

2. Sidusus. Seosed ühendavad kõiki sõnu omavahel.

3. Sobivus. Kõik sõnad lauses rahuldavad omavahel ühendumist piiravaid kitsendusi.

Lause võib olla LG jaoks ka mitmene – juhul, kui leidub rohkem kui üks tingimusi täitev seoste hulk. Sellise olukorra tekitab näiteks asjaolu, kus ühte sõna on võimalik tõlgendada nii verbi kui noomenina ja mõlemal puhul on seostetingimused täidetud.
Näide:

Toodud näites esinevad D-, S- ning O-tüüpi seosed. Igaüks ühendab parajasti kahte sõna:



1. D-seos ühendab artikleid nimisõnadega.

2. S-seos ühendab nimisõnu finiitsete verbidega.

3. O-seos ühendab transitiivseid verbe laienditega.

Seose tüüpi täpsustavad väiketähed. Näiteks s ja p näitavad nimisõna-verbi ühildumist (vastavalt ainsus ja mitmus), c märgib loendatavust.

LG olulisem komponent on sõnastik (*Dictionary*), mis peale sõnade esitab ka sõnade ühildumist kirjeldavad kitsendused. Reeglistikus on esitatud sõna koos seostega, mida ta võib luua kas vasakule (-) või paremale (+). Näiteks nimisõna *cat* tohib olla D-seose paremal pool ja S-seose vasakul pool, aga mitte O-seose vasakul pool. Tabelis on sõnadele antud ka seose esinemise tingimu-

Sõna	Seosed
a the	D+
ran	S-
Mary John	O- or S+
chased	S- & O+
snake cat	D- & (O- or S+)

sed, samuti on tähistatud see, kas mingi seose esinemine on kohustuslik või mitte. *LG*-s on suuremad sõnagrupid koondatud nn. *words*-failidesse, seoste komplekt antakse kogu faili kohta. Nii näiteks moodustavad omaette failid transitiivsed verbid, intransitiivsed verbid, loendatavad nimisõnad, loendumatud nimisõnad. Antud formalismi raames töötava süsteemi grammatika kirjeldamise ja loomise lihtsus lubab teda kasutada reaalsete süsteemide prototüüpide loomiseks.

Konverteerimine tähendab siin iga antud *LG*-lause puuks teisendamist, kus sõnade kui lehtede kohale on ehitatud vastav seoste hierarhia. Antud juhul on tegemist konverteerimise näitega, kus vormiliselt lihtsa teisenduse taga peitub väga keerukas (lingvistiline) töötlus.

1.3 Konverterite loomise automatiseerimisest

1.3.1 Probleemi püstitus

Üha suurenev vajadus erinevat tüüpi konverterite järele on tõstatanud loomuliku küsimuse, kuidas automatiseerida konverterite loomist. Muidugi on soovitav luua võimalikult üldist tüüpi konvertereid. Kõige ideaalsem oleks üks universaalne konverter tüüpi *Any-to-Any*. Nii üldisel kujul püstitatud ülesanne oleks ilmselt lahendamatu, kuid peale probleemi mõistlikku kitsendamist saab astuda olulisi reaalseid samme konverteriarenduse automatiseerimiseks.

Probleemi võib sõnastada järgmiselt: on antud lähtekeel, sihtkeel ja konverteerimise mudel, ülesandeks on automaatselt genereerida vastav konverter.

Käesolevas aga piirdume tegelikult veelgi enam lihtsustatud ülesandega – sellisega, kus sihtkeel on fikseeritud. Osutub, et sel juhul polegi vaja iga lähtekeele jaoks oma konverterit. Lähtekeelte (grammatikate) teatavale eeltöötlusele tuginedes saab piirduda vaid ühe konverteriga, mis vastavalt mudelile teisendab suvalise lähtekeelse teksti vajaliku kujuga sihtkeelseks tekstiks.

Käesolevas töös välja töötatud lähenemisviis on küll piiratud sihtkeele fikseeritusega, kuid teiselt poolt iseloomustab seda universaalsus lähtekeeelte pere osas: ainsaks nõudeks lähtekeelele on selle formaalse grammatika olemasolu.

1.3.2 Grammatika-põhise konverteriarenduse printsiip

Olgu SK fikseeritud sihtkeel ja $LK_1, LK_2, \ldots, LK_n$ mingi lähtekeelte pere, mille jaoks on tarvis konvertereid $LK_i \rightarrow SK (i = 1, 2, \ldots, n)$.

Põhimõtteliselt tuleb kõne alla (suhteliselt ebaefektiivne) tee: iga i ja mudeli M_i korral $(i = 1, 2, \ldots, n)$ luua (programmeerida) konverter $convert_i()$:

$convert_i(LK_i tekst) \rightarrow SK tekst$

või siis pisut üldisem $convert'_i()$, kus antakse lähtekeelne tekst koos mudeliga:

$convert'_i(LK_i tekst, M_i) \rightarrow SK tekst.$

Nii või teisiti, loodavad konverterid peavad sisaldama ka keele LK_i tekstide süntaksanalüüsi moodulit. Tegelikult oleks praktikas ka vajalik ühe baaskeele korral käsitleda mitte ühte vaid mitmeid varieeritud mudeleid.

Käesolevas töös välja töötatud konverteriarenduse süsteemi aluseks on idee, et tarvilik on vaid üks, teatavas mõttes universaalne konverter. Nimelt, kui fikseerida antud lähtekeelte pere korral süntaksanalüüsi puude (ehk parsipuude) esitamise keel PK, siis ainsaks tarvilikuks konverteriks osutub konverter tüüpi $PK \rightarrow SK$, mis suvalise parsipuu teisendab sihtkeelseks tekstiks vastavalt etteantud mudelile. Mudel on loomulikult sõltuv konkreetsest lähtekeelest (LK_i), esitades konkreetse konverteerimise reeglistiku. Iga lähtekeele LK_i jaoks peab olema loodud vastav süntaksanalüsaator $Parser_i()$:

$Parser_i(LK_i tekst) \rightarrow parsipuu.$

Lakooniliselt võib siis kirjeldatud konverteerimise protsessi esitada kujul
- - - $t \in LK_i$
$parsipuu := Parser_i(t)$
$t' := converter(parsipuu, M)$
- - - $t' \in SK$; t' on teksti t konverteerimise tulemus vastavalt mudelile M

Niisuguse lähenemise eelised on järgmised.

1. Lähtekeelte pere on avatud. Uue lähtekeele kasutusele võtmiseks on vaja luua vaid selle keele süntaksanalüsaator ehk parser. Viimane tegevus on automatiseeritav, nõudes ainult grammatika piisavalt täpset kirjeldamist.

2. Võimalike lähtekeelte valik on väga lai, kuna ainukeseks nõudeks on formaalselt kirjeldatud grammatika olemasolu.

3. Uue lähtekeele lisamine ei nõua programmeerimistööd (konverteri programmeerimist). Vajalik on esitada vaid mudeli(te) kirjeldus(ed).

4. Mudelite kirjeldamine on lihtne, võimaldades kasutajal hõlpsasti katsetada paljude mudelivariantidega ühe ja sama lähtekeele korral.

Uue lähtekeele lisamisel osutub kõige töömahukamaks lähtekeele grammatika viimine parseri genereerimiseks nõutavale kujule. Selle juures võib lihtkasutajal tarvis minna vastava spetsialisti abi.

Peatükk 2

Süsteem *Amadeus* kui konverteriarenduse platvorm

2.1 Süsteemi *Amadeus* üldiseloomustus

Amadeus [1][3] on *Java* keelel põhinev mitmevaateline, mitmekeelne, üldotstarbeline süsteem arvutitekstide[1], näiteks programmitekstide, LaTeX dokumentide, *HTML* ja *XML* tekstide ning teiste tarkvaradokumentide käsitlemiseks. Projekti *Amadeus* algseks eesmärgiks oli nii tekstilise kui ka graafilise toimeti ühendamine programmkoodi käsitlemiseks. Uuematesse versioonidesse on lisandunud mitmeid üldisemaid võimalusi.

Amadeus toetab arvutitekstide graafilist esitamist ja nende toimetamist. Toimetatavad tekstid esitatakse nn. skeemtekstidena, milles primitiivsemad (lihttekstilised) osad on ühendatud nn. skeemidesse. Skeemid kuvatakse kasutajaekraanile graafiliste konstruktsioonidena. Viimaste tegeliku graafilise "väljanägemise" määrab parajasti kasutatav vaade (mis on kasutaja poolt valitav). Süsteemi graafilisus tähendab eeskätt seda, et on võimalik toimetada mitte ainult lihttekstilist osa, vaid ka suuremaid terviklikke graafilisi konstruktsioone. Niisiis, käsitletava teksti jämestruktuur kirjeldatakse plokkidena (skeemidena), mis sisaldavad primitiividena ka tekstilisi elemente, nagu lihtsad andmed, kommentaarid, lihtsamad laused, avaldised ja tingimused.

[1] Arvutiteksti (tarkvaradokumendi) all mõistame sellist teksti-põhist infot, mida käsitlevad nii inimesed kui ka arvutid.

Amadeus töötab nii *Windows* kui ka *Unix* platvormidel. Suhtlemiseks saab valida inglise, eesti või vene keele. Järgnevates näidetes kasutatakse siiski inglisekeelseid menüüvalikute nimetusi.

2.2 Skeemteksti mudel ja skeemtöötlus

Skeemtöötluse eesmärgiks on viia süntaktiliselt erinevad arvutitekstid ühisele alusele – üldisele skeemmudelile. Skeemteksti mudel annab teatava metoodika arvutiteksti struktuuri konstrueerimiseks, kasutades lihtsamaid tekstiüksusi elementaarsete ehitusplokkidena. Sel moel konstrueeritud teksti nimetatakse *skeemtekstiks*. Suvaline skeemtekst saadakse tekstiüksuste korraldamisel puukujuliseks struktuuriks, mida nimetatakse *skeempuuks* ning milles teksti lihtsamad osad esinevad puu tippude tekst-atribuutidena.

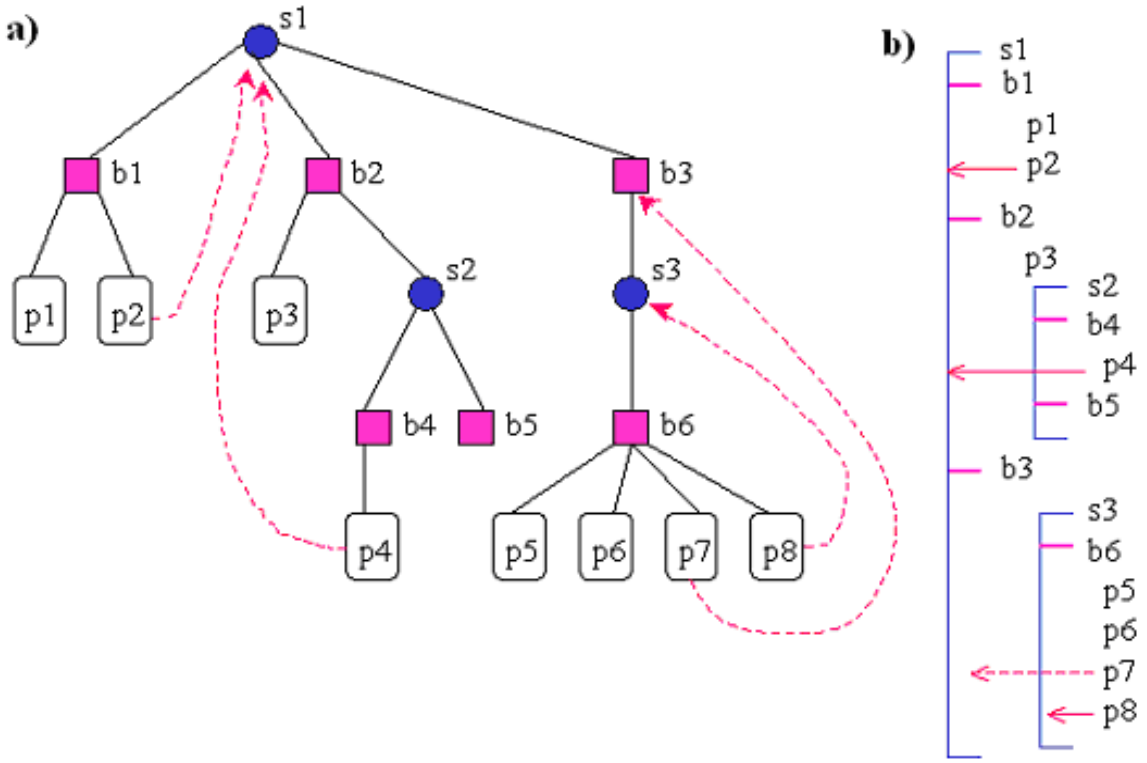
Täpsemalt, *skeempuu* on paar $<T,A>$, kus A on noolte hulk ja T on mittetühi puu järgmiste omadustega:

- on olemas kolme liiki tippe: skeemtipp ehk skeem (*sketch*), harutipp ehk haru (*branch*) ja primitiivitipp ehk primitiiv (*primitive*);
- T juureks on skeem;
- skeemi järglasteks võivad olla harud, harude järglasteks on primitiivid või skeemid ning primitiividel ei ole järglasi;
- igal skeemil on päis (*head*) – primitiivpäiste (*primitive head*) ehk elementaarpäiste järjend (erijuhul tühi).

Ühe haruga skeemi nimetatakse *monoskeemiks*.

Joonisel 2.1 on toodud näide, kuidas võiks välja näha abstraktne skeempuu ning vastava struktuuriga skeemtekst. Kõrvuti tavapärase puuvaatega (a) on kasutatud skeemvaadet (b). Viimane on intuitiivselt küllaltki hästi arusaadav ja ka toimetamise seisukohast ning paigutuse poolest mugavam kui puuvaade.

Süsteemis *Amadeus* esitatakse jooksvad arvutitekstid skeemtekstidena, kusjuures algteksti lihtosad on skeemteksti puu tippude atribuutideks. Need



Joonis 2.1: Skeempuu graafiline esitus (a) ja esitus skeemides (b).

tekstiüksused ei ole piiratud puhttekstilise kujuga, põhimõtteliselt võimaldatakse ka piltide või omakorda skeemtekstide kasutamist reasümbolitena.

Joonis 2.2 kirjeldab BNF (*Backus-Nauri* notatsiooni) abil skeemteksti täpsemat süntaksit.

Skeemteksti vaade (kuvakuju) ei ole jäigalt fikseeritud: seades vastava atribuudi, saab skeemtekstile omistada suvalise vaate olemasolevate vaadete hulgast. Süsteemis *Amadeus* on skeemtekst *WYSIWYG*-toimetatav igas vaates. Kõrvuti tavalise teksti toimetamisteisendustega (primitiivide jaoks) toetab *Amadeus* ka toimetamistegevusi skeemide ja primitiivide käsitlemiseks (lisamine, kustutamine, tüübi muutmine jne.) tervikuna.

Kuvatava teksti baaskeel määratakse skeemtekstide korral vastava atribuudi abil. *Baaskeel* on keel, mille grammatika kirjeldab skeemtekstile vastavat lihteksti.

```
skeemtekst :: = skeem »
kavand :: = skeem | haru»
skeem :: = skeemi_keha kavandi_atribuudid»
skeemi_keha :: = (haru)+»
haru :: = haru_keha kavandi_atribuudid»
haru_keha :: = (liige)*»
liige :: = skeem | primitiivliige»
primitiivliige :: = primitiiv | nool»
primitiiv :: = tüüp primitiivtekst»
nool :: = primitiiv lõputase»
lõputase :: = non-negative_integer»
tüüp :: = non-negative_integer»
primitiivtekst :: = (rida)* kommentaar»
kommentaar :: = rida»
rida :: = (rea_element)+»
rea_element :: = sümbol | hüpersümbol»
sümbol :: = character font color»
hüpersümbol :: = pilt | hüperelement»
pilt :: = ikoniseeritus image»
hüperelement :: = skeemtekst»
kavandi_atribuudid :: = tüüp kommentaar päis
ikoniseeritus view base_language »
päis :: = (primitiiv_päis)*»
primitiiv_päis :: = primitiiv»
ikoniseeritus :: = true| false»
```

Joonis 2.2: Skeemteksti süntaksi kirjeldus.

Võrreldes tavatoimetitega pakub süsteem *Amadeus* arvutitekstide mugavama, skeemkujul toimetamise võimaluse. Muuseas on loomuliku funktsioonina kasutatav ka teksti voltimine (*code folding* – programmide korral). Suvalist skeemi saab vabalt valida (hiire vasakpoolsele nupule klõpsates) ja ikoniseeri-

da/deikoniseerida (hiire parempoolsele nupule klõpsates). Samuti on lubatud valitud skeemi aktiivses aknas skeemistada, tekstualiseerida või redutseerida, kasutades vastavaid meetodeid, mis sõltuvad valitud baaskeelest.

Käesolevas vaadeldava konverteerimisküsimuse seisukohalt on väga oluline, et süsteemile *Amadeus* on seesmiselt omane tekstide teisendamise protsess. Tegelikult kuulub süsteemi mitte üks skeemisti, vaid iga aktsepteeritava baaskeele jaoks leidub vastav spetsiifiline skeemisti. Sama kehtib ka tekstualiseerijate kohta (vt. joonis 2.3). Süsteemi *Amadeus* kui multitoimeti seisukohalt on skeemistid ja tekstualiseerijad hädavajalikud vahendid "lihttekstide maailmaga" sideme pidamiseks.

Nii skeemistid kui ka tekstualiseerijad on sisuliselt eriotstarbelised konverterid. Selles mõttes võib süsteemi *Amadeus* vaadelda teatava konverteerimisplatvormina. Näiteks saame konverteri keelest $L1$ keelde $L2$, kui lisame süsteemi *Amadeus* skeemisti

$L1 \rightarrow \{skeemtekst\}$

ja tekstualiseerija

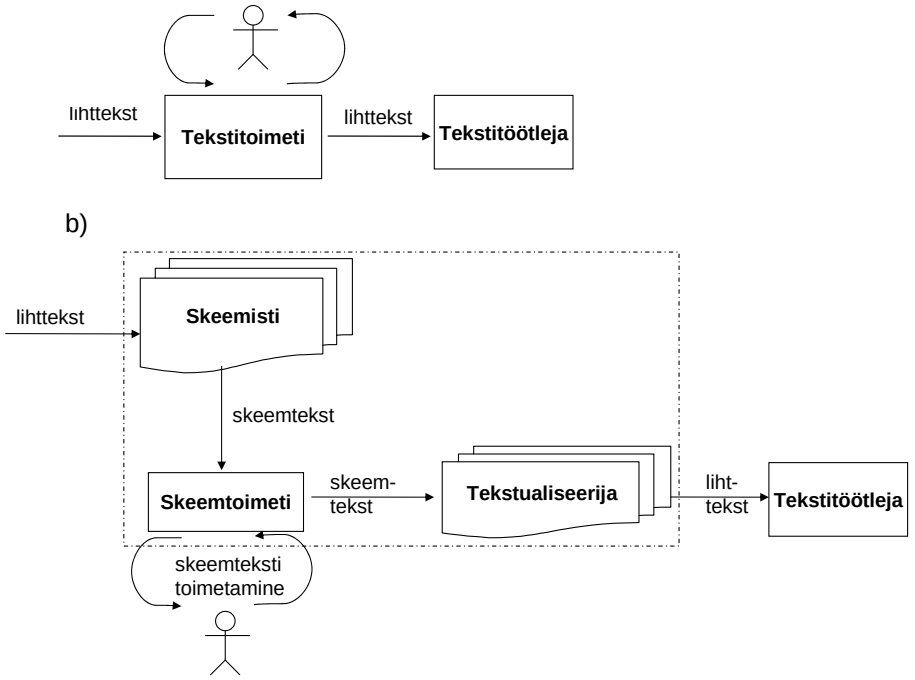
$\{skeemtekst\} \rightarrow L2$.

Lisaks säilib mugav võimalus toimetada "vahekuju" – skeemteksti.

2.3 Skeemistamine ja tekstualiseerimine

Süsteemi *Amadeus* kõik need funktsioonid, mis sõltuvad jooksva (toimetatava) skeemteksti baaskeelest, on koondatud menüüvaliku *Tools* alla. Alamvalikute loetelu on järgmine:

- Skeemistamine (*Tools + Sketchify*) – jooksva lihtteksti (reastruktuuriga teksti) viimine skeemkujule ja tulemuse kuvamine uues aknas.
- Parsimine (*Tools + Parse*) – jooksva lihtteksti parsimine; tulemuseks on parsipuu skeemkujul (uues aknas).
- Tekstualiseerimine (*Tools + Textualize*) – jooksva skeemteksti teisendamine lihttekstiks (uues aknas).
- Redutseerimine (*Tools + Reduce*) – jooksva skeemteksti eriteisendus jooksvas aknas.



Joonis 2.3: Teksti töötlemine tavatoimetiga (a) ja skeemtoimetiga (b).

- Normaliseerimine (*Tools + Normalize*) – jooksva skeemteksti eriteisendus uude aknasse.
- Trükivalmistus (*Tools + Prepare TeX*) – jooksva skeemteksti ettevalmistamine (uude aknasse) TEXvõi LATEX abil printimiseks, peamiselt erisümbolite asendamine vastavate TEX-fragmentidega.

Funktsioonide ülaltoodud selgitused on pisut lihtsustatud: üldiselt teisendatakse jooksva skeemteksti (hiireklõpsuga) valitud osa, selgitustes on aga eeldatud, et valitud on kogu jooksev skeemtekst. Parsimise funktsioon on lisatud *Amadeus* lähteversioonile käesoleva töö käigus. Eriteisenduste tähendus ei ole fikseeritud, nende spetsiifikat võib iga baaskeele puhul suvaliselt määrata.

Märgime, et jooksev lihttekst kujutub *Amadeus* aknas monoskeemina, mille ainukese haru kõik liikmed on (kommenteerimata tekstiga) primitiivid. Lihttekst sisestatakse tavalisest tekstifailist menüükäsuga *File + Read text* ja salvestatakse tavafaili menüükäsuga *File + Write text*.

Süsteemis *Amadeus* realiseeritakse iga baaskeel *Java* klassina, mis pärineb ülemklassist *BaseLanguage* ja realiseerib muuhulgas selle (abstraktsed)

meetodid $sketchify()$ ning $textualize()$. Nimetatud meetodid rakenduvad vastavate menüüvalikute käivitamisel, nende poolt sooritatavad tegevused on aga sisuliselt konverteerimised.

Kui nt. klass *BaseLanguageLKi* realiseerib baaskeele LK_i, siis selle meetod $sketchify()$ on konverter tüüpi $LK_i \rightarrow SK$, kus SK on skeemtekstide esitamise keel, ja meetod $textualize()$ on konverter tüüpi $SK \rightarrow LK_i$. Mõlemate puhul on konverteerimise mudel nö. sisse ehitatud (fikseeritud).

Süsteemi *Amadeus* lähteversioonis (nimetusega *AmadeusJS*) olid juba realiseeritud nt. keeled *Java*, *HTML*, *LATEX*, mille konversioonimudelid on täpsemalt kirjeldatud artiklis [2]. Nende keelte konverterid loodi nö. käsitsi, seejuures eriti töömahukaks osutus skeemistamise meetodi ($sketchify$) realiseerimine. Käesolevas töös keskendutaksegi just skeemistuskonverterite loomise automatiseerimisele – küsimusele, kuidas uue baaskeele LK korral võimalikult efektiivselt välja töötada vastav skeemistamise meetod, mis realiseeriks konverteri tüüpi $LK \rightarrow SK$. Vastavate automatiseerimisvahenditega täiendatud süsteemi *Amadeus* uus versioon (nimetusega *AmadeusJSN*) kujutab endast juba küllaltki efektiivset konverteriarenduse tarkvara.

Peatükk 3

Grammatika-põhine skeemisti loomine

Käesolevas peatükis kirjeldatakse üldist põhimõtet, kuidas süsteemis *Amadeus* mingi antud lähtekeele LK jaoks luuakse skeemisti, st. konverter keelest LK skeemkeelde SK, lähtudes keele LK grammatikast.

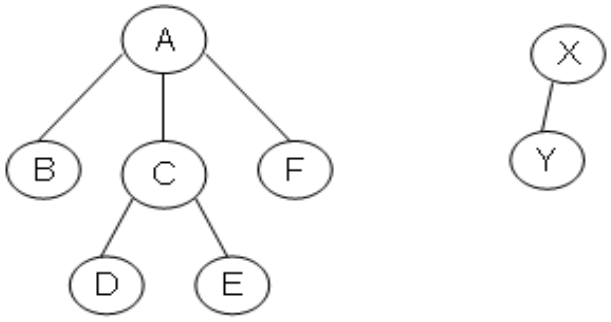
3.1 Näiteülesanne

Skeemisti loomise protsessi illustreerimiseks kasutame lähtekeele rollis väga lihtsat puu/metsa esitamise keelt VSE. Iga puu või mets kirjutatakse selles keeles nn. vasaku suluesitusena [5]. Tippude nimedeks on identifikaatorid, eraldajateks aga ümarsulud ning komad.
Metsa vasak suluesitus on

– tühi, kui mets on tühi,
– vastasel korral aga ümarsulgudesse võetud metsa puude vasakute suluesituste loetelu (komadega eraldatult).

Mittetühja puu vasak suluesitus koosneb juurest ja selle järgnevast juure alampuude metsa vasakust suluesitusest.
Näiteks `(A(B,C(D,E),F),X(Y))` on metsa vasak suluesitus, mille graafiline esitus skeemvaates on toodud joonisel 3.1.

Ülesandeks on **luua skeemisti – konverter tüüpi $VSE \rightarrow SK$, mis suvalise metsa vasaku suluesituse teisendab skeemkujule**.



Joonis 3.1: Metsa graafiline esitus.

Vaadeldud näitemetsa korral võiks skeemistamise soovitud tulemuseks olla skeemtekst joonisel 3.2. Teisendamise reegel ehk konverteerimise mudel (sõnalises vormis) olgu järgmine:
metsa iga puu T kujutatakse ühe haruga moodulskeemina[1], mille päises asub puu juur ja mille haru liikmeteks on puu T alampuid kujutavad skeemid.

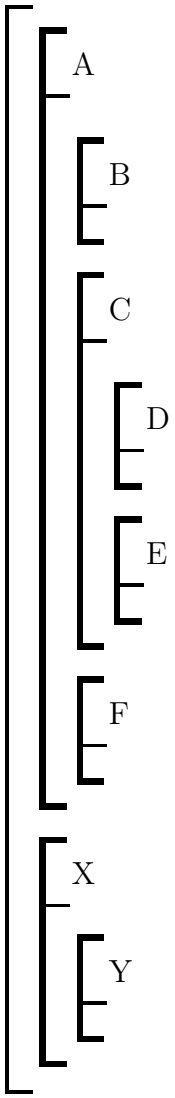
Märgime, et joonisel 3.2 kujutatud struktuuriga skeemteksti saab süsteemis *Amadeus* hõlpsasti kuvada ka joonisel 3.3 näidatud viisil. Selleks tuleb vaid vaateks valida (*View + Set*) puuvaade $STree$.

3.2 Grammatika esmane spetsifitseerimine

Kui lähtekeel on valitud ja konverteerimise eemärk esialgse mudelina selgeks tehtud, järgneb lähtekeele grammatika täpsustamine. Eesmärgiks on koostada grammatika selline kirjeldus, mille alusel saaks automaatselt genereerida lähtekeele parseri. Parseri ülesandeks on lähtekeele tekstide jaoks analüüsipuude ehk parsipuude ehitamine. Kuna parseri genereerimiseks kasutatakse süsteemi *JJTree/JavaCC*, siis tuleb grammatika lõppude lõpuks esitada *JJTree* [8] sisendtekstina. Süsteem *JJTree/JavaCC* ja selle kasutusvõimalused süsteemi *Amadeus* kontekstis leiavad üksikasjalikku valgustamist A. Reitsaka magistritöös [6].

Grammatika spetsifitseerimise aluseks võetakse reeglina lähtekeele BNF-

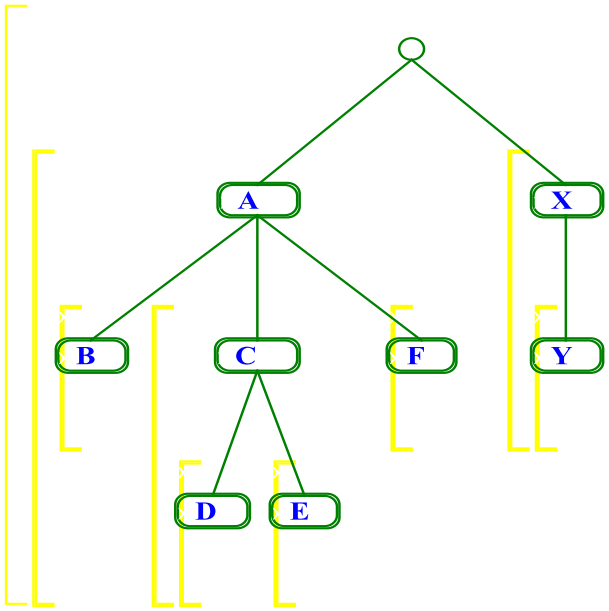
[1]Erinevalt lihtskeemist kuvatakse moodulskeem paksemat skeemijoont kasutades.



Joonis 3.2: Metsa esitus skeemtekstina (skeemvaates).

grammatika[2]. Tõsi küll, üksikutel juhtudel õnnestub alustada paremast stardipaigast – juba *JavaCC* vormis olevast grammatikast. Nt. mõningate tuntud keelte (enamuses – programmeerimiskeelte) *JavaCC*-grammatikad leiduvad

[2]Tegelikult kasutatakse küll BNF laiendatud vormi, kus on lubatud nt. meta-avaldised `( )*`, `( )+`, `( )?`, [ ].



Joonis 3.3: Metsa esitus skeemtekstina (puuvaates).

vastavas repositooriumis [7] ja neid saab kasutada alusena vajaliku *JJTree*-grammatika väljatöötamisel või siis eeskujuna (struktuurilt repositooriumist saaduga sarnaneva lähtekeele korral).

Näitena vaadeldava lähtekeele VSE üldine BNF-grammatika on muidugi väga lihtne:

```
s ::= metsaVSE
metsaVSE ::= "(" puuVSE ("," puuVSE)* ")"
puuVSE ::= juur (metsaVSE)?
juur ::= IDENTIFIER
```

See spetsifikatsioon täpsustab muuhulgas ka tühja metsa küsimuse. Antud grammatika (teise produktsiooni) kohaselt ei saa mets olla tühi, selles peab leiduma vähemalt üks puu.

3.3 Grammatika spetsifitseerimine *JJTree* sisendi kujul

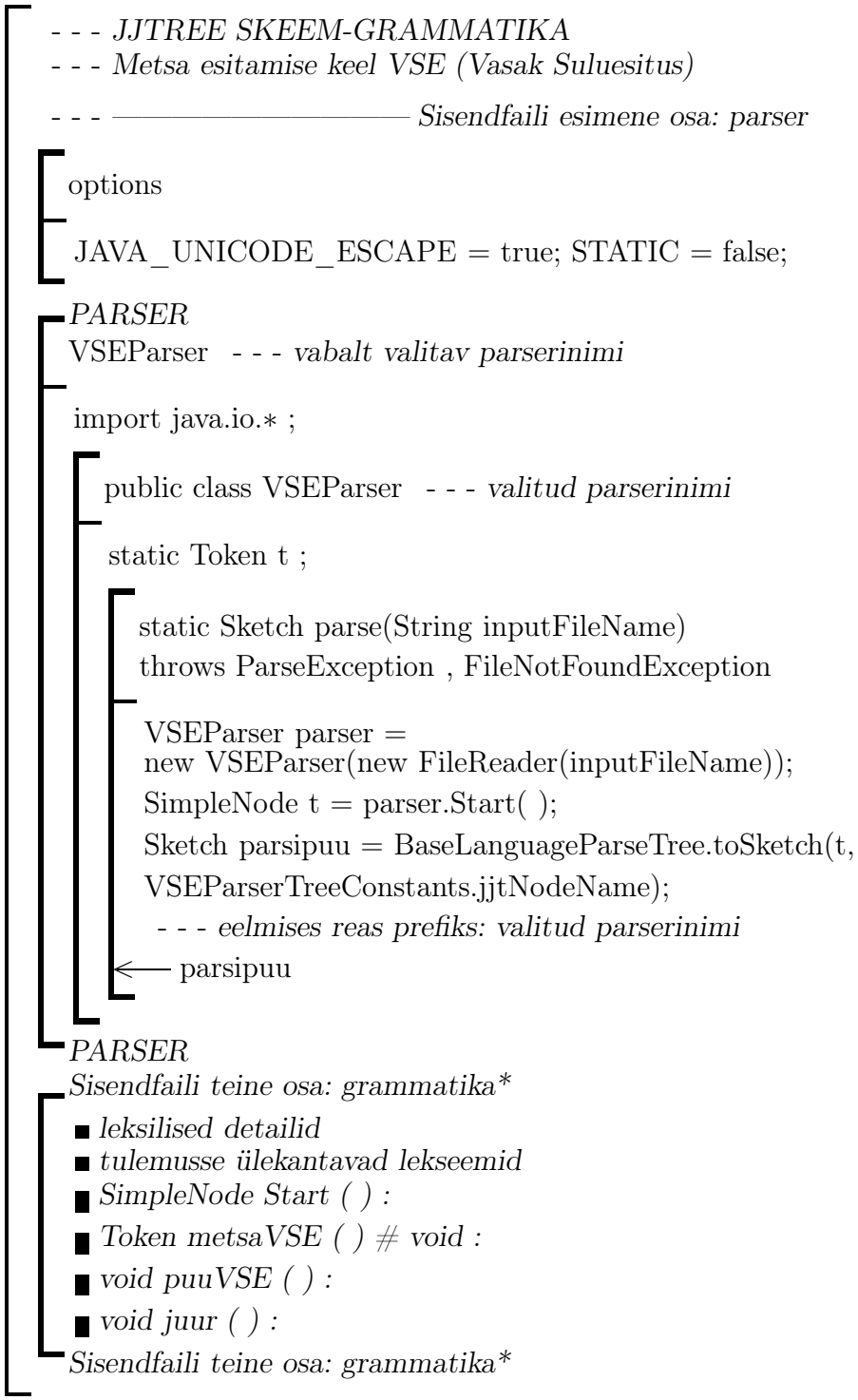
Grammatika spetsifitseeritakse sellisel kujul tekstifailina – nn. grammatikafailina, millest töötlusvahendite *JJTree* ja *JavaCC* järjestikusel rakendamisel, saadakse parser lähtekeele jaoks.

Loodav grammmatikafail koosneb kahest osast. Esimene osa on standardne (vaadeldavas kontekstis) ja kujutab endast parseri töö üldist *Java* vormis esitatud algoritmi. Teine osa on grammatika kirjeldus *JTTree/JavaCC* vormis [6], mis struktuurilt on samuti väga lähedane *Java*-tekstide keelele. Teiselt poolt, lähtegrammatikas sisalduvad produktsioonid esituvad grammatikafailis küllaltki *BNF*-produktsioonidega sarnaneval kujul.

Järgnevalt (lk. 32 - 35) on esitatud meie näitekeele *VSE* grammatikafail (skeemkujul). Tänu sellele, et süsteemi *Amadeus* on juba integreeritud vastav baaskeel (nimetusega *JavaCC*), ei pruugi grammatikafaile arendada lihttekstidena. Grammatikafaile saab toimetada süsteemis *Amadeus* ülevaatlikumal skeemkujul ja hõlpsasti väljastada lihttekstidena nende edasiseks kasutamiseks töötlussüstemis *JTTree/JavaCC*. Sama grammatikafail lihttekstiks teisendatuna leidub Lisas 1.

Grammatikafaili esimene osa (skeemid *options* ja *PARSER*, lk. 32) on standardne ning selle võib iga uue lähtekeele jaoks lihtsalt kopeerida. Muuta tuleb (kolmes kohas) vaid parserinimi. Ka teise osa alguses määratletud leksilised detailid, nagu tühemikud, eraldajad ja identifikaatorid, on tavaliselt üsna sarnased siinses näites toodutele.

Küll aga tuleb konverteerimisel tulemusse ülekantavate lekseemide jaoks kirjeldada lisaproduktsioonid. Antud juhul on ülekantavaks vaid üks lekseem, nimelt juure nimi – identifikaator. Iga ülekantava lekseemi (siin näites: `<IDENTIFIER>`) jaoks tuleb kasutusele võtta uus mitteterminal (siin on selleks võetud `tIdentifier`) ja kirjeldada vastav produktsioon (siin: skeem päisega `void tIdentifier`). Algsetes grammatika produktsioonides tuleb iga sellise ülekantava lekseemi (siin: `<IDENTIFIER>`) asemel kirjutada vastav mitteterminal (siin `tIdentifier()`, vt. viimane skeem päisega `void juur`, lk. 35).


- - - JJTREE SKEEM-GRAMMATIKA
- - - Metsa esitamise keel VSE (Vasak Suluesitus)
- - - ———————— Sisendfaili esimene osa: parser
options
JAVA_UNICODE_ESCAPE = true; STATIC = false;
PARSER
VSEParser - - - vabalt valitav parserinimi
import java.io.* ;
public class VSEParser - - - valitud parserinimi
static Token t ;
static Sketch parse(String inputFileName)
throws ParseException , FileNotFoundException
VSEParser parser =
new VSEParser(new FileReader(inputFileName));
SimpleNode t = parser.Start( );
Sketch parsipuu = BaseLanguageParseTree.toSketch(t,
VSEParserTreeConstants.jjtNodeName);
- - - eelmises reas prefiks: valitud parserinimi
⟵ parsipuu
PARSER
Sisendfaili teine osa: grammatika*
leksilised detailid
tulemusse ülekantavad lekseemid
SimpleNode Start ( ) :
Token metsaVSE ( ) # void :
void puuVSE ( ) :
void juur ( ) :
Sisendfaili teine osa: grammatika*

leksilised detailid

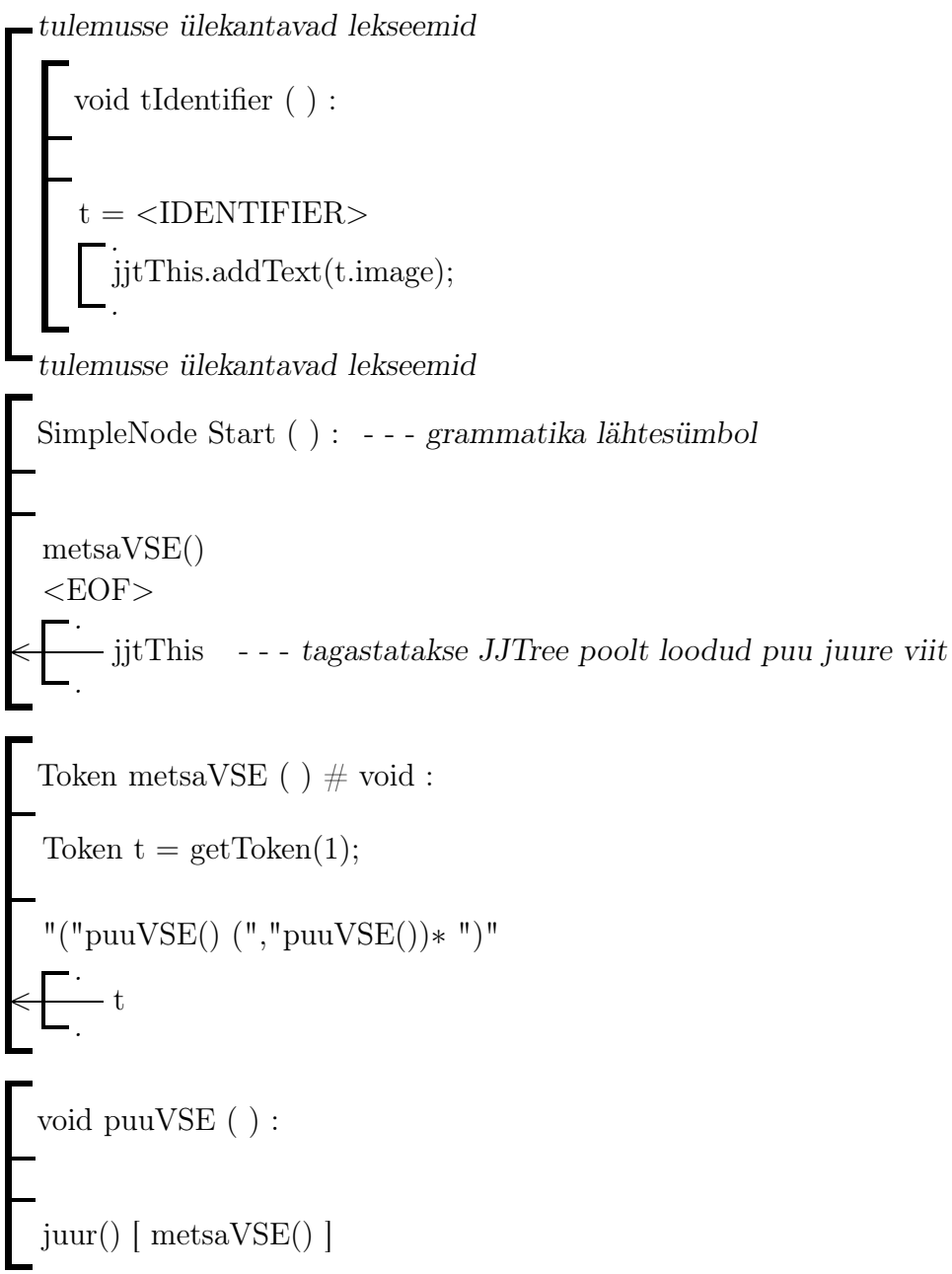
SPECIAL_TOKEN:
tühemikud
" " | "\t"| "\n"| "\r"| "\f"

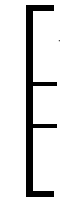
TOKEN :
identifikaator
< IDENTIFIER : < LETTER > (< LETTER> | <DIGIT >)* >
|
< # LETTER : ["A"− "Z", "_", "a"− "z"] >
|
< # DIGIT : ["0"− "9"] >

TOKEN :
eraldajad
< LPAREN : "(">
|
< RPAREN : ")">
|
< COMMA : ",">

TOKEN:
muu
< ANYTHING_ELSE: (~[ ]) >

leksilised detailid


tulemusse ülekantavad lekseemid
void tIdentifier ( ) :
t = <IDENTIFIER>
jjtThis.addText(t.image);
tulemusse ülekantavad lekseemid
SimpleNode Start ( ) : - - - grammatika lähtesümbol
metsaVSE()
<EOF>
jjtThis - - - tagastatakse JJTree poolt loodud puu juure viit
Token metsaVSE ( ) # void :
Token t = getToken(1);
"("puuVSE() (","puuVSE())* ")"
t
void puuVSE ( ) :
juur() [metsaVSE()]



Grammatika sisuline osa kirjeldatakse *produktsiooniskeemidena* (antud juhul neli viimast skeemi), millest kaks esimest on üsnagi standardsed, tagastades vastavalt `SimpleNode` ja `Token` tüüpi väärtuse. Ülejäänud produktsioonide tagastustüübid määratakse vastavalt vajadusele. Lihtsamates grammatikates need produktsioonid reeglina midagi ei tagasta. Ka vaadeldava näite kahes viimases produktsioonis on tagastustüübiks ainult tühitüüp `void`.

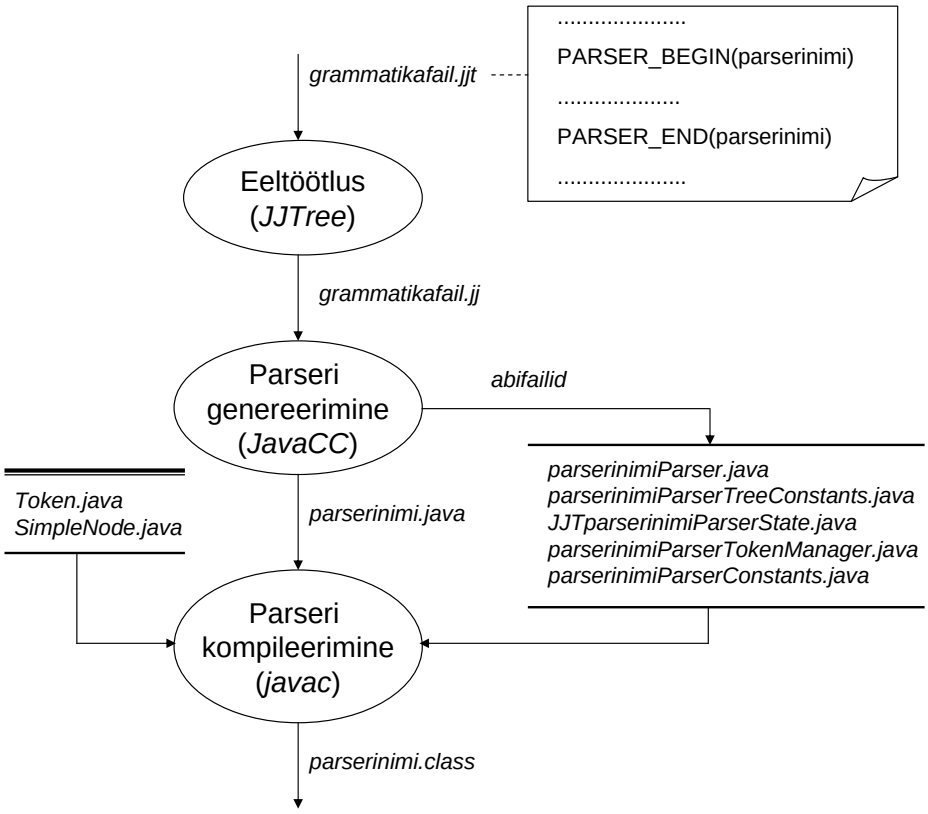
Koostatud grammatikafail silutakse: vastav lihttekstifail (laiendiga `.jjt`) töödeldakse *JJTree/JavaCC/javac* abil (vt. joonis 3.4). Kui selle töötluse käigus ilmneb vigu, siis korrigeeritakse grammatikafail ja töödeldakse uuesti. Õnnestunud töötluse tulemusena saadaksegi antud lähtekeele parser. Vaadeldava näite korral esitub parser *Java* klassina *VSEParser.class*.

Selleks, et parserit kasutada lähtekeele tekstide parsimiseks ja skeemistamiseks tuleb esiteks süsteemile *Amadeus* lisada lähtekeelele vastav baaskeel. Nimelt luuakse grammatikafailis kirjeldatud parseri töö (vt. joonis 3.5) tulemusena parsitava lähteteksti parsipuu skeemkujul. Selle edasine käitlemine toimub juba süsteemi *Amadeus* vahenditega.

Parsipuu struktuuri selgitatakse järgmise jaotise lõpus. Märgime vaid, et parsipuusse ei tule nendele mitteterminalidele vastavaid tippe, mille skeemproduktsiooni päisele *JJTree* grammatikas on lisatud piiritleja `#void`. Antud näitegrammatikas on elimineeritavaks kuulutatud seega mitteterminal `metsaVSE`.

3.4 Uue baaskeele lisamine

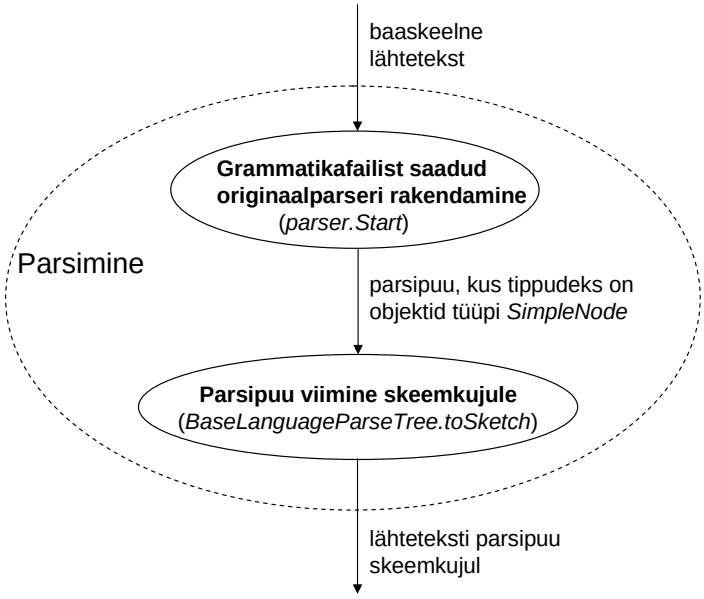
Baaskeele lisamiseks luuakse uus baaskeele klass ja korrigeeritakse baaskeelte ühine ülemklass *BaseLanguage*. Lähtekeelele *LK* vastava uue baaskeele klassi nimi koostatakse kujul *BaseLanguage<LK>*. Seega siin vaadeldava lihtsa näitekeele puhul oleks baaskeele klassi nimeks *BaseLanguageVSE*.



Joonis 3.4: Parseri valmistamine *JJTree* vormis grammatika alusel.

Joonisel 3.6 on esitatud mingit baaskeelt realiseeriva klassi üldine ülesehitus. Baaskeele klassi loomine toimub järkjärgult. Lähtekohaks on vastav mall (vt. Lisa 2). Esiteks realiseeritakse ainult baaskeele klassi parseri osa, jättes teised osad (*SKETCHIFY*, *REDUCE* ...) tühjaks. Seega mallist baaskeele klassi esimese versiooni saamiseks tuleb malli tekstis teha vaid neli asendust. Näiteks keele *VSE* baaskeele klassi saame mallist, asendades seal neljas kohas "`LK`" → "`VSE`".

Uue baaskeele süsteemi integreerimiseks tuleb modifitseerida ülemklass. Lisas 3 esitatud klassi *BaseLanguage* struktuuris on välja toodud kõik täienda-



Joonis 3.5: Grammatikafailis kirjeldatud parseri andmevoo diagramm.

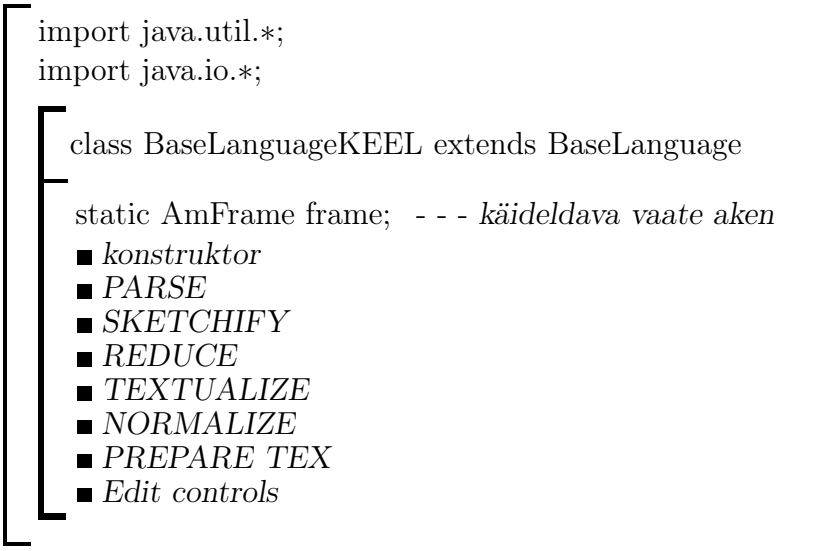
mist vajavad kohad (baaskeele *VSE* lisamise näitel). Vajaduse korral saab meetodisse *prepareSourceTextFile* lülitada veel lähtetekstide täiendava eeltöötluse.

Kui ülemklass (*BaseLanguage*) on modifitseeritud ja uue baaskeele klassi (nt. *BaseLanguageVSE*) esimene versioon ülalkirjeldatud viisil loodud, siis saab süsteemi *Amadeus* juba kasutada parseri edasisel sisulisel testimisel. Sisestanud (*File + Read text*) mingi lähtekeelse (*VSE*) lihtteksti, nt.

```
(A(B,C(D,E),F),X(Y))
```

saame selle baaskeeleks omistada (*Base + Set*) keele *VSE* ja rakendada parsimist (*Tools + Parse*). Tulemuseks on antud teksti parsipuu (vt. joonis 3.9) uues aknas.

Järgnevalt toimub baaskeele klassi edasine täiustamine, selle järgmiste osade välja arendamine. Konverteriarenduse seisukohalt on kõige olulisem skeemistamise osa (*SKETCHIFY*, joonis 3.6) realiseerimine. Viimasega seo-



Joonis 3.6: Baaskeelt (KEEL) realiseeriva klassi struktuur.

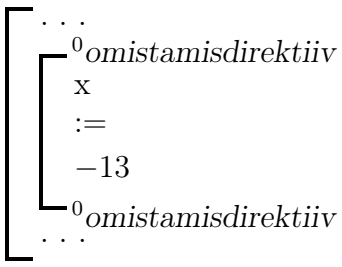
tud küsimused leiavad käsitlemist järgmises jaotises.

Antud lähtekeele parsimistegevus (käivitatakse vastava baaskeele klassi osas *PARSER*) seisneb grammatikafaili alusel saadud parseri rakendamises ja tulemuse skeempuuks teisendamises (vt. joonis 3.5). Skeempuuks teisendamise sooritab meetod *toSketch* klassis *BaseLanguageParseTree*. Viimane on eriotstarbeline baaskeele klass, milles on realiseeritud osa *REDUCE* , kus leiduv meetod *reduce* võimaldab lihtsustada skeemkujul parsipuud.

Parsipuu skeemkujus on iga mitteterminal (parsipuu sõlm ehk sisetipp) esitatud skeemina, mis sisaldab kõiki temale alluvaid mitteterminale (st. neid kujutavaid alamskeeme). Mitteterminali nimi varustatakse prefiksiga ^{0} ja paigutatakse skeemi kommentaari kohale. Antud näites (joonisel 3.9) esindavad mitteterminale skeemid kommentaaridega ^{0}*puuVSE* ja ^{0}*juur*.

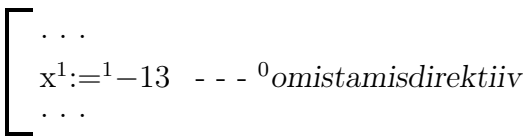
Terminalid (parsipuu lehed) on kujutatud primitiividena, mille tekstiks on vastav lekseem. Antud näites esinevad terminalid tekstidega "A", "B", "C", "D", "E", "F", "X", "Y".

Mitteterminali, millel pole mitteterminaalseid järglasi, s.t. on ainult terminaalsed järglased, nimetatakse *primitiivseks mitteterminaliks*. Primitiivse mitteterminali (nimega *omistamisdirektiiv*) näide leidub joonisel 3.7.



Joonis 3.7: Primitiivne mitteterminal parsipuus.

Lähteteksti parsimise vahetuks tulemuseks on redutseerimata parsipuu (baaskeelega *Parse tree*) *Amadeus* aknas. Sellele saab rakendada (menüükäsuga *Tools + Parse*) redutseerimise funktsiooni – käivitada ülalmainitud redutseerimise meetod. Redutseerimine seisneb primitiivsete mitteterminalide skeemide lihtsustamises järgmisel viisil. Iga primitiivse mitteterminali skeem parsipuus asendatakse primitiivliikmega, mille kommentaariks on antud skeemi kommentaar (^{0} ja mitteterminali nimi) ning mille tekstiks on skeemis esinevate primitiivliikmete tekstide järjend (sümboliga ^{1} eraldatult). Joonisel 3.7 kujutatud skeempuu redutseeritakse kujule joonisel 3.8. Näitena vaadeldava keele *VSE* tekstide parsipuudes on kõigil primitiivsetel mitteterminalidel (^{0}*juur*) parajasti üks alluv, seega konkreetse näiteteksti parsipuu (joonisel 3.9) redutseerimise tulemuses (vt. joonis 3.10) eraldajat ^{1} vaja ei lähe.



Joonis 3.8: Redutseeritud primitiivne mitteterminal.

3.5 Skeemkonversiooni mudel

Pärast lähtekeele (LK) jaoks uue baaskeele klassi parsimise osa valmimist ja testimist piisava arvu lähtetekstide parsimise katsetamise teel, on süsteem juba praktiliselt valmis töötama skeemkonverterina $LK \rightarrow SK$. Tuleb vaid pisut täiendada süsteemi programmikoodi ja kirjeldada soovitava skeemkonversiooni mudel. Baaskeele (joonis 3.6) skeemistamise ossa *SKETCHIFY* kopeeritakse mall (joonis 3.11), milles muudetakse ainult üks rida. Selles fikseeritakse faili nimi, kust süsteem leiab skeemkonversiooni (skeemistamise) mudeli. Vajaduse korral saab skeemistamise osa meetodisse *postprocessResult* lülitada veel skeemistamise tulemuse täiendava järeltöötluse.

Järgnevas tutvustatakse lähemalt skeemkonversiooni mudeli mõistet.

Skeemkonversiooni mudel (edaspidi: mudel) süsteemis *Amadeus* kujutab endast sisuliselt parsipuu skeemtekstiks teisendamise eeskirja. Mudeliga määratud teisenduse teostab klassi *BaseLanguage* meetod

$sketchifyParseTree(Sketch\ s,\ Sketch\ m)$,

mis teisendab antud parsipuu s skeemtekstiks, tehes seda vastavalt antud mudelile m. Sisuliselt teostab meetod konversioone tüüpi $LK \rightarrow SK$: kui s on keele LK mingi lähteteksti t parsipuu, siis tulemuseks on skeemtekst $s' \in SK$ – skeemkujule konverteeritud lähtetekst t. Meetod eeldab, et mudel ise on esitatud skeemkujul. Väga oluline on meetodi töö sõltumatus konverteeritavast lähtekeelest: meetod on muutmatul kujul rakendatav suvalise baaskeele tekstide parsipuude teisendamiseks.

Lähtekeele LK mudeliks süsteemis *Amadeus* on skeemtekst, milles leiduvaid skeeme ja primitiive kasutatakse mallidena keele LK parsipuude teisendamisel sihtskeemtekstideks. Eeldatakse, et teisendatavad parsipuud on esitatud redutseeritud kujul (redutseerimise tähendust selgitati eelmises jaotises). Mudeli ja parsipuu elemendid seostatakse mitteterminalide nimede kaudu. Nii nagu parsipuus, nii ka mudelis on mitteterminalide nimed varustatud prefiksiga ^{0}.

Üldiselt, kui mitteterminal ^0mt leidub mudelis mingi skeemi SM kommentaaris, siis parsipuu iga skeem kommentaariga ^0mt (skeem SP) konverteeritakse skeemiks, millel on samad attribuudid nagu skeemil SM. Skeemi SP selliselt muudetavateks (skeemist SM pärinevateks) atribuutideks on skeemi

tüüp, baaskeel, ikoniseeritus ja skeemi kommentaar. Viimane tähendab seda, et skeemi SP kommentaariks omistatakse skeemi SM kommentaari teksti algusosa, mis eelneb mitteterminali nimele ${}^0mt$. Nt. kui mudelis leidub skeem kommentaariga `abc`^{0}`Key`, siis iga skeem kommentaariga ^{0}`Key` parsipuus konverteeritakse skeemiks kommentaariga `abc`.

Peale skeemide konverteeritakse ka parsipuus esinevad primitiivsed mitteterminalid. Nagu selgitatud eelmises jaotises, esitub primitiivne mitterminal ${}^0pmt$ skeemi liikmena kujul

`lehttippude`^{1}`lekseemide`^{1}`järjend - - -` ${}^0pmt$.

Taolise liikme PP konverteerimise määrab mudelis leiduv primitiivliige (või primitiivpäis) PM, mille tekstis sisaldub selle mitteterminali nimi ${}^0pmt$. Nimelt omistatakse liikmele PP primitiivi PM tüüp ning primitiivi tekstis asendatakse eraldajad ^{1} tühikutega. Kui konverteerimist määrav PM asub mudelis mingi skeemi päises, st. on primitiivpäiseks, siis primitiiv PP paigutatakse konverteeritavas parsipuus ümber lähima hõlmava skeemi päisesse. Primitiivpäise teksti ette lisatakse veel primitiivi PM teksti algusosa, mis eelneb mitteterminali nimele ${}^0pmt$, ja lõppu lisatakse primitiivi PM teksti lõpuosa, mis järgneb mitteterminali nimele ${}^0pmt$.

Mudel joonisel 3.12 kirjeldab käesoleva peatüki algul kirjeldatud konversiooni $VSE \rightarrow SK$, eeldusel, et grammatika on spetsifitseeritud ja uus baaskeel lisatud eelmistes jaotistes kirjeldatud viisil. Kui see mudel on koostatud ja salvestatud, siis suvalise *VSE*-keelse teksti (mingi konkreetse metsa vasaku suluesituse) konverteerimiseks nõutud skeemkujule tuleb see tekst vaid sisestada (*File + Read text*), omistada talle baaskeeleks *VSE* (*Base + Set*) ja skeemistada jooksev tekst (*Tools + Sketchify*). Tulemus kuvatakse uues aknas.

Parsipuu ülal esitatud teisendusmoodus lubab mudelit ehitada üsna mitmel moel. Tähtis on vaid see, et mudelis esinevad skeemid ja primitiivid määraksid soovitud konversiooni. Muidugi on loomulik püüda mudel esitada võimalikult informatiivsena, näidates selles võimalikke skeemide (ja primitiivide) tüüpilisi vastastikuseid asendeid/sisalduvust. Lisaks on lubatud mudelis iga mitteterminali nime lõppu lisada veel tärn, mis konverteerimist kuidagi ei mõjuta. Seda võimalust saab kasutada näitamaks mõnede mudelielementide (vastete) võimalikku kordumist sihttekstis.

Mudeliga joonisel 3.12 samaväärne, kuid informatiivsem mudel leidub joonisel 3.13. Võimalik konstruktsioonide kordumine sihttekstis on näidatud tärniga vastava mitteterminaali nime lõpus.

Esialgse mudelis võib hõlpsasti saada mingist konkreetsest redutseeritud parsipuust, milles leiduvad kõik mitteterminalid. Selleks, on vaja lihtsalt muuta parsipuud (muuta see mudeliks) ja salvestada nagu süsteemi *Amadeus* suvaline teine skeemtekst.

Mudelite koostamine on lihtne, eriti võrreldes nt. erikonverterite programmeerimisega või modifitseerimisega. Seetõttu on väga kerge katsetada mitmeid sisuliselt erinevaid mudeleid.

Joonisel 3.14 on esitatud mudel metsa vasaku suluesituse konverteerimiseks skeem-*XML* kujule. Selles leiduvad ka tulemusse kanduvad päise tekstifragmendid. Mudelis on skeemide baaskeeleks määratud *XML*. Viimane on üks süsteemi *Amadeus* varem integreeritud baaskeeli. Näiteteksti konverteerimise tulemus selle mudeli järgi on toodud joonisel 3.15. Paneme ka tähele, et erinevalt eelmistest konverteerimise näidetest, esituvad nüüd puud (selle mudeli kohaselt) lihtskeemidena, mitte moodulskeemidena. Antud juhul on tegemist konverteerimisega tüüpi $VSE \rightarrow skeem\text{-}XML$.

Kuna baaskeelele *XML* vastav baaskeele klass realiseerib ka konverteerimise tüüpi $skeem\text{-}XML \rightarrow XML$ (st. tekstualiseerimise), siis kokkuvõttes on nüüd süsteem *Amadeus* kasutatav ka konverterina tüüpi $VSE \rightarrow XML$. Nt. saab konverteerimise tulemusele joonisel 3.15 rakendada *XML*-tekstualiseerimist (*Base + Set ... XML* ja *Tools + Textualize*), mis annab tulemuseks *XML* teksti joonisel 3.16.

3.6 Kokkuvõte

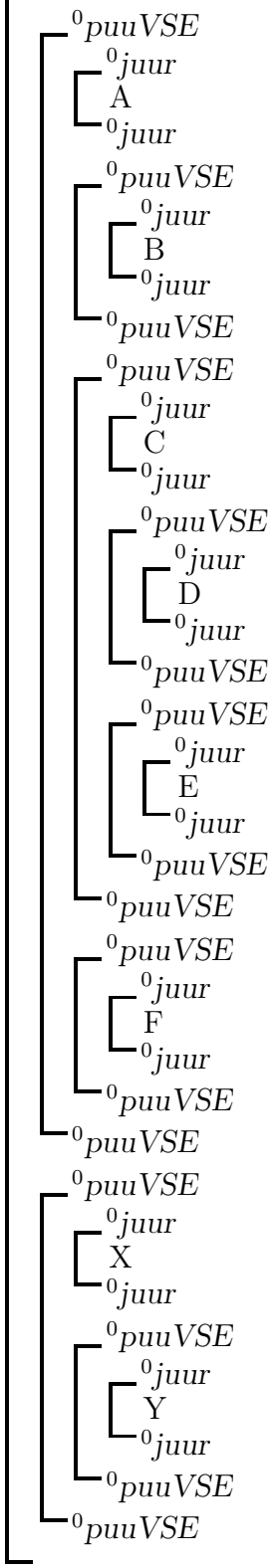
Skeemkonverteri arendamiseks süsteemi *Amadeus* kasutades tuleb spetsifitseerida lähtekeele grammatika ja integreerida vastav uus baaskeel. Viimases tuleb realiseerida vähemalt parsimise ning skeemistamise osad. Skeemkonverteri kasutamiseks peab eelnevalt veel konkretiseerima konversiooni mudeli.

Taolist moodust vahetult kasutades on koostöös juhendaga J. Kiho loodud konverter *JJTree/JavaCC* kujul kirjutatud lihttekstiliste grammatikafailide

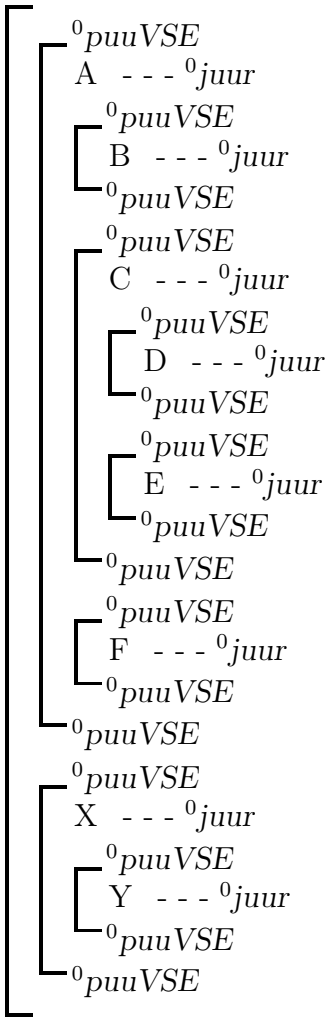
viimiseks skeemkujule. Vastavalt lisati uus baaskeel nimega *JavaCC*. Parserit kirjeldava grammatikafaili struktuur on esitatud Lisas 7, konversioonimudel aga Lisas 8. Grammatikafail arendati välja lihtteksti vormis, lähtudes repositooriumis [7] leiduvast grammatikast. Selle skeemkuju (Lisas 7) on aga juba saadud loodud konverteri rakendamise tulemusena.

Praktilist skeemkonverteri arendusprotsessi iseloomustab iteratiivsus. Nimelt ei õnnestu (ega pole ka vajalik) kohe luua lõplikult vastuvõetav grammatikafail. Esimese sammuna luuakse esialgne "töötav" grammatika, mida hakatakse järkjärgult modifitseerima sihiga saada selliste omadustega parsipuud, mille jaoks osutub lihtsaks nende soovitud skeemkujule teisendamise kirjeldamine, st. konverteerimismudeli spetsifitseerimine. Peamisteks muudatusteks grammatikas on ebavajalike mitteterminalide parsipuudesse ülekandumise keelamine. Nagu eespool selgitatud, reguleeritakse seda lisapiiritlejaga `#void` skeemproduktsioonide päistes. Märgime, et tegelikult osutub iteratiivseks ka mudeli loomine: sobivaima mudeli leidmiseks tuleb katsetada mitmete erinevate mudelivariantidega.

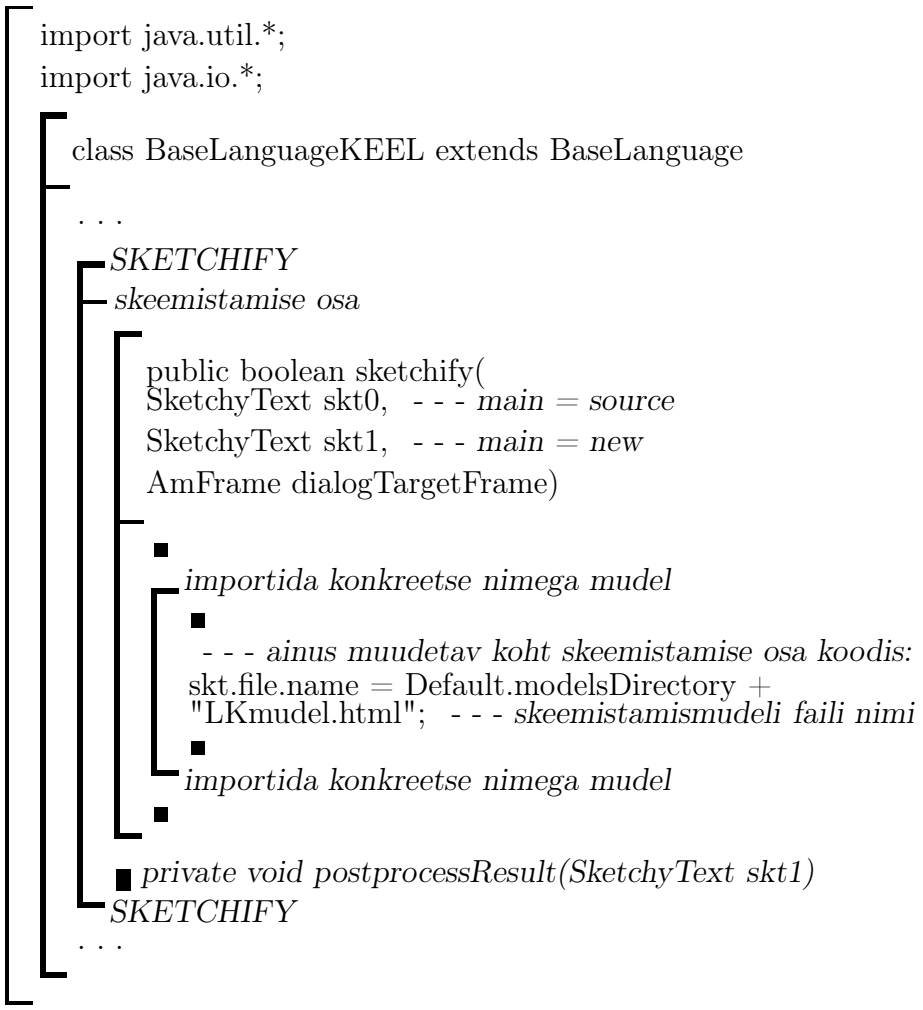
Grammatika sobivale kujule viimine on protsessi sisuline, loomingulisem külg. Sellega kaasneb aga üsnagi palju rutiinset tööd. Ka algatuslikus osas (esialgse grammatika loomine ja uue baaskeele lisamine) tuleb arendajal kokkuvõttes teha (ja korrata) rohkesti lihtsaid, tehnilisi üksiktegevusi. Konverteriarendusel vajalike, kuid rutiinsete tehnilist laadi tööde automatiseerimise küsimustele ongi pühendatud järgmine peatükk.



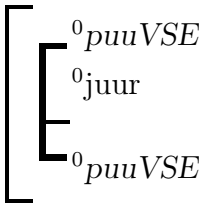
Joonis 3.9: VSE baaskeele näiteteksti `(A(B,C(D,E),F),X(Y))` parsipuu.



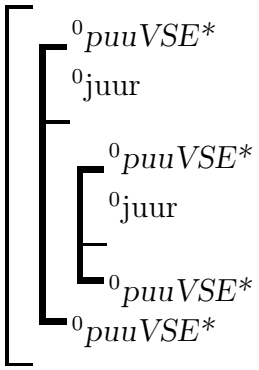
Joonis 3.10: VSE baaskeele näiteteksti `(A(B,C(D,E),F),X(Y))` redutseeritud parsipuu.



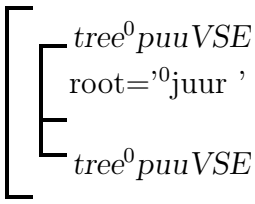
Joonis 3.11: Baaskeele klassi skeemistamise osa mall.



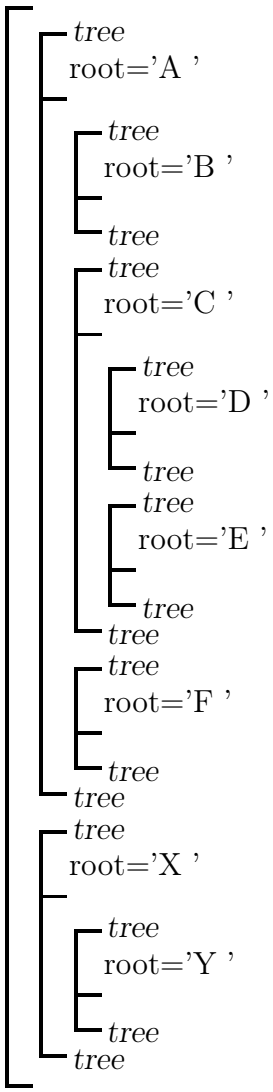
Joonis 3.12: *VSE* skeemkonversiooni mudel (I).



Joonis 3.13: *VSE* skeemkonversiooni mudel (II).



Joonis 3.14: *VSE* skeemkonversiooni mudel (III).



Joonis 3.15: *VSE* näiteteksti konverteerimise tulemus mudeli III (joonis 3.14) järgi.

```
<tree root='A '>
 <tree root='B '>
 </tree>
 <tree root='C '>
  <tree root='D '>
  </tree>
  <tree root='E '>
  </tree>
 </tree>
 <tree root='F '>
 </tree>
</tree>
<tree root='X '>
 <tree root='Y '>
 </tree>
</tree>
```

Joonis 3.16: *VSE* näiteteksti *XML* kujule konverteerimise tulemus.

Peatükk 4

Baaskeelte käitlemise ja skeemisti loomise automatiseerimine

Käesoleva töö praktiliseks tulemiks on süsteemi *Amadeus* uus versioon *AmadeusJSN*. Viimane on saadud lähteversioonist *AmadeusJS* konverteriloomet toetavate erisuste lisamise teel. Automatiseeritud on baaskeeltega manipuleerimine (uue baaskeele lisamine, olemasoleva kustutamine, kustutatud baaskeele taastamine). Grammatikafaili iteratiivse arendamise toeks on spetsifitseeritud vastav projektikirjeldus.

Antud peatükis esitatakse ülevaade peamistest süsteemile *AmadeusJSN* omastest, uude versiooni lisandunud tehnilistest lahendustest.

4.1 Süsteemi *Amadeus* kataloogi ülesehitus

Kuna süsteemis leidub väga palju erinevat liiki ja erineva otstarbega faile, siis on loomulik need grupeerida mitmetesse kataloogidesse (ehk kaustadesse). Süsteemi kodukataloogil on kindel ülesehitus, mille üldine struktuur on toodud joonisel 4.1.

Alamkataloogide otstarve on järgmine:

1. ***class***: objektide klassid (*Java* klassifailid), samuti *Amadeus.properties* fail (määrab süsteemi sätingud algkäivitusel, lähemalt vt. allpool) ja

Joonis 4.1: Süsteemi *AmadeusJSN* kataloogi struktuur.

(soovitavalt) eriotstarbeline standardne fail *amadeus.tex* – trükimakrod skeemtekstide TEX/LATEX kujul väljastamiseks;

2. ***docHtml***: süsteemi enda lähtekood, *Amadeus* salvestusformaadis (*html* või *xml*) failidena;

3. ***gif***: süsteemi poolt kasutatavad pildid (*collapsed.gif* – ikoniseeritud skeemi pildi fail);

4. ***grammars***: *JJTree* kujul grammatikafailid (laiendiga *.jjt*); skeemkujul grammatikafaile (laiendiga *html* või *xml*) hoitakse selle kataloogi alamkataloogis ***grammarsSK***;

5. ***java***: süsteemi lähtekoodi *java*-tekstid (tekstualiseeritud kataloogist ***docHtml***) ja baaskeelte grammatikafailid *JavaCC* kujul laiendiga *.jj* (saadud kataloogi ***grammars*** failidest);

6. ***JavaCC***: alamsüsteem *JJTree/JavaCC*;

7. ***macros***: toimetamismakrod (*edit-macros* [1]). Makrode kasutamine lihtsustab kasutaja tööd, sest makrode failis saab defineerida lühendeid tihti kasutatavate skeemide, tekstide jms. tarbeks.

8. ***models***: grammatikate/keelte spetsiifilised skeemkonversiooni mudelid *Amadeus* salvestusformaadis (*html*) failidena.

9. ***patterns***: koodiosade ŝabloonid süsteemi *Amadeus* arendajale; nt. tsüklid skeempuu läbimiseks jmt.;

10. ***Personal***: kasutaja isiklik kataloog; peamine otstarve hetkel on kasutaja poolt kustutatud baaskeelega seotud failide säilitamine alamkataloogis ***BL_ repository*** (baaskeele võimalikuks taastamiseks edaspidi);
    alamkataloogi ***BL_ repository*** otstarve:
    baaskeele kustutamisel (jaotis 4.4) luuakse selles alamkataloogis vastava baaskeele nimeline alamkataloog, milles luuakse omakorda järgmised alamkataloogid:

    (a) *docHtml:* baaskeele klassi jaoks

    (b) *grammars:* baaskeele grammatikafaili jaoks

    (c) *Portfolio:* baaskeele projektikirjelduse jaoks

11. ***Portfolio***: projektikirjeldused[1] *Amadeus* salvestusformaadis (*html*) failidena;

12. ***temp***: suvalised ajutised failid (varukoopiad jmt.).

4.1.1 Algsätingute fail

Kõige esimene asi, millest peab kasutaja alustama, kui hakkab süsteemiga *Amadeus* töötama, on vajalike süsteemsete parameetrite sätestamine. Korrastada tuleb algsätingute fail *Amadeus.properties*, mis asub reeglina süsteemi *class* kataloogis. Konkreetne näide (reanumbrid on lisatud kommenteerimise huvides):

[1]Projekti mõistet selgitatakse jaotises 4.2

(1) `AmadeusDirectory=D:\\AmadeusJSN\\`
(2) `iconImageFileName=D:\\AmadeusJSN\\gif\\collapsed.gif`
(3) `modelsDirectory=D:\\AmadeusJSN\\models`
(4) `JavaJSDK=C:\\j2sdk1.4.2_05\\bin`
(5) `Backup=D:\\AmadeusJSN\\BackupN\\`
(6) `defaultFont=Times New Roman, 1, 12`
(7) `defaultOpenDirectory=D:\\AmadeusJSN\\Portfolio`

Selles failis peab kasutaja korrigeerima kataloogiteed süsteemi *Amadeus-JSN* kodukataloogini (1), ikoniseeritud skeemi pildi failini (2), mudelite kataloogini (3), *Java* kodukataloogini (4), varukoopiate kataloogini (5) ning lisaks veel määrama ekraanil kuvatavate skeemtekstide kirjastiili ja kirja suuruse (6). Võimalik on määrata ka kataloog (7), millest soovitakse alustada avatava (projekti) faili otsimist.

4.2 Projekti mõiste süsteemis *Amadeus*

Projekt süsteemis *Amadeus* on failinimede hulk, kus iga failinimega on seotud toimetamise eel- ja järeltegevuste sooritamiseks vajalikud andmed ehk atribuudid [13][14]. Projektiga grupeeritud faile nimetatakse projekti failideks.

Atribuudid jagunevad kolme liiki – *Paths*, *Open-script* ja *Save-script* atribuudid:

- *Paths*: projekti failide kataloogiteed. *Path* liiki atribuutide peamine eesmärk on anda lühinimed füüsilistele kataloogiteedele muutmaks projekti failidele viitamise sõltumatumaks konkreetsest keskkonnast.
- *Open-script*: tegevused/funktsioonid, mida rakendatakse failidele nende avamisel. Reeglina on üheks tegevuseks projekti faili importimine (avamine).
- *Save-script*: tegevused/funktsioonid, mida rakendatakse failidele nende salvestamisel. Reeglina on üheks tegevuseks projekti faili eksportimine (salvestamine).

Projektiga (projekti all) töötamisel kuvatakse iga (avatud) projekti fail eraldi spetsiaalses aknas, kus *File* menüüs on aktiivsed ka valikud *Open* ja *Save* (tavaaknas, st. mitte projekti faili aknas pole need valikud aktiivsed). Menüükäsk *File+Open* sellises aknas initsieerib tegevuste jada, mis on defineeritud selle projekti *Open*-skriptis. Menüü käsk *File+Save* initsieerib tegevuste jada, mis on defineeritud projekti *Save*-skriptis.

Projekti enda struktuur ja sisu esitatakse skeemtekstina, mida nimetatakse *projektikirjelduseks* (seda sisaldavat faili aga projektifailiks). Nõuded projektikirjelduse skeemtekstile on järgmised:

1. Projektikirjeldus esitatakse moodulskeemina, millel on üks elementaarpäis ja kaks haru.

2. Projektis võib olla alamprojekte. Alamprojekti kanduvad pärimise teel ülemprojekti kõik atribuudid, mis alamprojektis pole määratud.

3. Projekti failinimed peavad olema unikaalsed kogu projektikirjelduses, kaasaarvatud alamprojektide projektikirjeldused.

4. Projektikirjelduse elementaarpäise tekstiks on projekti nimi; seda ei tohi tühjaks jätta. Projekti nimi kuvatakse iga projekti faili akna ülaservas.

5. Projektikirjelduse esimeses harus asub 0 või enam projekti atribuudirühma. Projekti *atribuudirühm* kujutatakse ühe elementaarpäisega üheharulise lihtskeemina.

Projektikirjelduse skeemteksti struktuuri näide on toodud joonisel 4.2.

projektikirjeldus (moodulskeem kahe haruga)

1.haru; attribuudid

Paths - - - *path-atribuutide grupp*
märgis: väärtuste paar
- - - *üldine kuju:*
L:X - - - *L - kasutaja poolt valitud märgis; X - väärtus*
- - - *näited:*
DOC:d:\dir
JAVA:d:\myJava
Backup:c:\temp
JSDK:C:\"Program Files"\Java\j2sdk1.5.0\bin\

Open-script - - - *avamise skripti atribuutide grupp*
tegevus: parameetrite paar
- - - *näide: importida(avada) fail kataloogist d:\dir*
Import: path=DOC - - - *tegelikult: Import: path=d:\dir*

Save-script - - - *salvestamise skripti atribuutide grupp*
tegevus: parameetrite paar
- - - *eksportida (salvestada) liht-HTML kujul faili d:\dir\jooksev.a:*
Export plain-HTML: path=DOC
- - - *tekstualiseerida ja tulemus kirjutada faili d:\myJava\jooksev.java:*
Textualize+Write text: path=JAVA ext=java
- - - *käivitada protsess javac käsurealt - javac jooksev.java:*
Process:{JDSK}javac path=JAVA ext=java

2. haru; projekti failide nimed, alamprojektide kirjeldused
fail1.a - - - *projekti fail*
fail2.a - - - *projekti fail*
▮ *alamprojekti kirjeldus*

projektikirjeldus (moodulskeem kahe haruga)

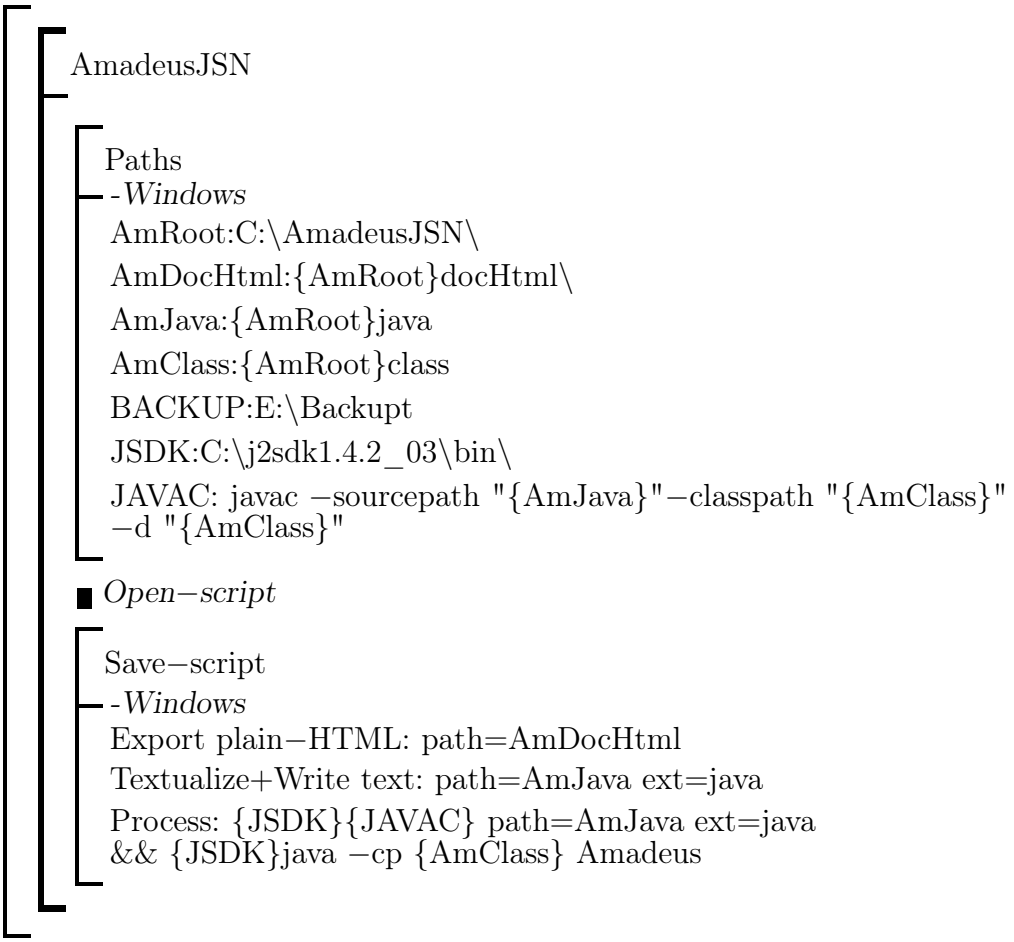
Joonis 4.2: Projektikirjelduse näide.

Põhioperatsioonid projektikirjeldusega ja projekti failidega:

1. Projektikirjelduse loomine/avamine: *File + Import Project* (avaneb aken valitud projektikirjeldusega ja tühja projektifaili aknaga).
2. Projektikirjelduse sulgemine: kasutada süsteemseid vahendeid.
3. Projektikirjelduse salvestamine: nagu tavalise skeemteksti salvestamine, st. *File + Export-plain HTML*.
4. Projekti failide avamine: hiirega kolmik-klõps või projektifaili akna menüüst valida *File + Open*.
5. Projekti faili sulgemine: kasutada süsteemseid vahendeid.
6. Projekti faili salvestamine: projekti faili akna menüüst valida: *File + Save*.

Atribuudirühmades *Open-script* ja *Save-script* näidatakse soovitavad tegevused menüükäskudena, mida igal avamisel ja salvestamisel tuleb sooritada. Olulise iseärasusena tuleb märkida võimalust *Save-script* rühmas kasutada *Process* funktsiooni (joonis 4.3). Selles kirjeldatakse käsurealt käivitavad (salvestamisjärgselt sooritamisele tulevad) operatsioonid. Kui käsureas on ette nähtud mitu järjestikust operatsiooni, siis on soovitatav need eraldada && abil (mitte aga eraldajaga |). Sellega tagatakse parem veadiagnostika (veaga lõppenud operatsiooni järel). Veateated väljastatakse *Amadeus* uues aknas. *Process* käsu projektikirjelduses defineerimisel ongi parameetriks käsurea tekst, kuid käsurea teksti defineerimisel saab kasutada *Path* rühma atribuute. *Process* käsu parameetrite *path* ja/või *ext* ilmumise kohale paigutatakse moodustatud faili nimi. Looksulud koos atribuudi märgendiga asendatakse vastava (*Path* rühma) atribuudi väärtusega.

Nt. olgu jooksva projekti faili nimeks *BaseLanguage* ja projektikirjelduses atribuudirühmad sellised, nagu näidatud joonisel 4.3. Siis selle jooksva faili sulgemisel sooritatakse ka *Save* rühma lõpus kirjeldatud *Process* käsk. *Path* rühmas määratletud atribuute arvestades tuleb aga tegelikult täitmisele järgmine käsurida:



Joonis 4.3: *Process* funktsiooni kasutamise näide.

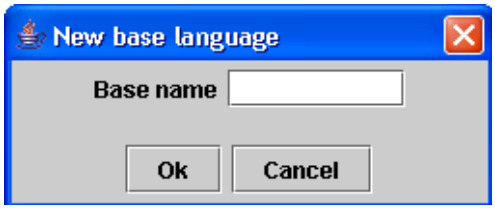
```
C:\j2sdk1.4.2_03\bin\javac
-sourcepath "C:\AmadeusJSN\java"
-classpath "C:\AmadeusJSN\class" -d "C:\AmadeusJSN\class"
C:\AmadeusJSN\java\BaseLanguage.java &&
C:\j2sdk1.4.2_03\bin\java -cp C:\AmadeusJSN\class Amadeus
```

Käesolevas vaadeldava skeemkonverteri loomise probleemi kontekstis on projektikirjeldus tähtis just arenduse iteratiivse osa jaoks. Jaotises 4.6 spetsifitseeritakse projekt grammatikafaili töötlemiseks, mille all on mugavam ja kiirem arendustöid teha.

4.3 Uue baaskeele automaatne lisamine

Süsteemi *Amadeus* on täiendatud uue funktsiooniga, mis käivitub menüüvalikuga *Base + Add new language*. Selle funktsionaalsuse realiseerimiseks on loodud klass *Item12Listener*, mille struktuur on toodud Lisas 4.

Baaskeele lisamiseks peab kasutaja avama uue *Amadeus* akna (*Frame + New*) ja selles uue skeemteksti (*File + New*), seejärel märgistama tekkinud skeemi ja valima *Base + Add new language*. Ilmub dialoogiaken, mis on toodud joonisel 4.4. Dialoogiaknasse sisestatakse lisatava baaskeele nimi.



Joonis 4.4: Uue baaskeele lisamise dialoogiaken

Nupu *OK* vajutamisel toimub esiteks uue baaskeele nime valideerimine, mille käigus kontrollitakse, kas antud nimega baaskeel on juba kasutusel või kunagi kasutusel olnud (passiivne). Kui antud nimega baaskeel on hetkel kasutusel, siis antakse kasutajale vastav veateade (*Antud nimega baaskeel juba eksisteerib!*). Kui aga antud nimega baaskeel ei ole kasutatavate keelte loetelus, kuid leidub kunagi kasutusel olnud baaskeelte loetelus, siis antakse kasutajale võimalus baaskeele taastamiseks (vt. jaotis 4.5). Pärast baaskeele (edukat) valideerimist toimub uue baaskeele automaatne lisamine, mille käigus sooritatakse järgmised tegevused.

1. Luuakse uue baaskeele projektikirjelduse fail, mis kirjeldab grammatikafaili töötlemise tegevusi (täpsemalt vt. jaotis 4.6). Projektikirjelduse faili nimeks on uue baaskeele nimi (laiendiga *.html*). Näiteks, kui kasutaja soovis lisada baaskeele nimega *ABC*, siis saab loodud projektikirjelduse faili nimeks *ABC.html*. Projektikirjelduse fail salvestatakse automaatselt *Amadeus* süsteemi kataloogi *Portfolio*. Projektikirjelduse kõik *Path* parameetrid hangitakse failist *Amadeus.properties*, nii et

kasutajal pole vaja midagi juurde lisada ega muuta.

Järgnevates tegevuste kirjeldustes eeldatakse, et lisatava baaskeele nimeks on *ABC*.

2. Luuakse uus klassi *BaseLanguage* alamklass *BaseLanguageABC* joonisel 3.6 esitatud malli kohaselt, tehes selles baaseele nimega seotud muudatused. Eeldatakse, et malli fail (*BaseLanguageKEEL.html*) paikneb kataloogis *docHtml*. Uus alamklass *BaseLanguageABC.html* salvestatakse samuti süsteemi kataloogi *docHtml*.

   Analoogilise protseduuriga valmistatakse ka vastav *java*-fail *BaseLanguageABC.java*, mis salvestatakse aga süsteemi kataloogi *java*.

3. Failis *BaseLanguage.html* (ja vastavalt samuti failis *BaseLanguage.java*) muudetatakse uue baaskeelega seotud parameetrid (vt. jaotis 3.4). Muudetavate kohtade täpsem nimekiri:

   - Süsteemis kasutatavad baaskeelte konstandid ja nende konstantide tähised. Siin esinevad ainult aktiivsete baaskeelte konstandid. Loetelu lõppu lisatakse järjekordne konstandi kirjeldus

     `final static byte ABC = a;`

     kus `a` on kõikide baaskeelte arv (koos lisatava baaskeelega).

   - Aktiivsete baaskeelte nimede loetelu, eelmisele (konstantide) loetelule vastavas järjekorras. Sõnedemassiivi

     `static String[ ] items`

     algväärtustuse lõppu lisatakse uus element `"ABC"`.

   - Dialoogis kuvatavate baaskeelte nimekirja kuuluvate keelte konstantide loetelu. Tavaliselt sisaldab kõigi aktiivsete keelte konstandid (kuid võib olla ka vaid osa nendest). Määrab, milliste aktiivsete keelte nimed dialoogis üldse kuvatakse ja mis järjekorras seda tehakse. Konstantidemassiivi

     `static byte[ ] currentSetOfBL`

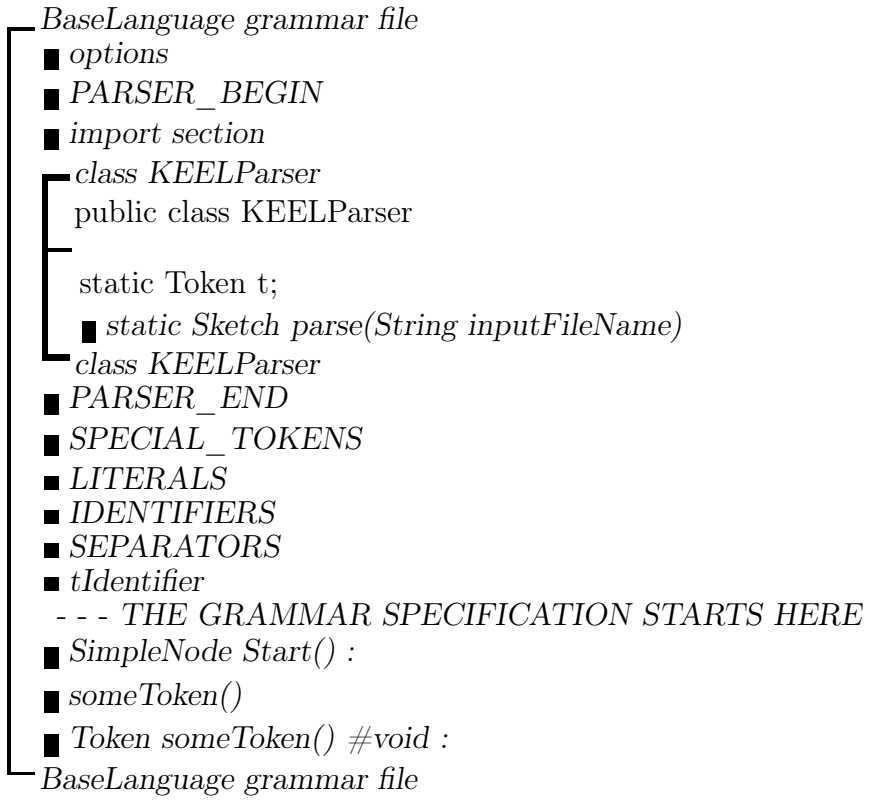
     algväärtustuse lõppu lisatakse uueks elemendiks `ABC`.

   - Kõikide, nii aktiivsete kui ka passiivsete baaskeelte nimede loetelu. Sõnedemassiivi

`static String[ ] useditems`
algväärtustuse lõppu lisatakse uus element `"ABC"`.

- Täiendus baaskeelte ülemklassis (*BaseLanguage*). Meetodisse
  `static BaseLanguage newBaseLanguage(byte type)`
  (vt. ka Lisa 3) lisatakse variant
  ```
  case ABC:
  return new BaseLanguageABC();
  ```

4. Luuakse uue baaskeele grammatikafaili lihttekstiline mall (laiendiga *.jjt*), mis salvestatakse süsteemi kataloogi *grammars*. Kui nt. lisatava baaskeele nimeks on *ABC*, siis luuakse fail *ABC.jjt*. Mall korrigeeritakse baaskeele nime osas, selle üldine struktuur on kujutatud joonisel 4.5.



Joonis 4.5: Grammatikafaili malli struktuur

5. Failide automaatne töötlemine (failinimede ette on veel lisatud süsteemi kataloogi nimetus).

   - Grammatikafaili teisendamine:
     $jjtree : grammars\backslash ABC.jjt \rightarrow java\backslash ABC.jj$
     Protsessori *jjtree* poolt genereeritakse (ja paigutatakse kataloogi *java*) ka failid
     ABCTokenManager.java,
     ABCTreeConstants.java,
     ABCParserConstants.java,
     JJTABCParserState.java
   - Parseri lähtefaili genereerimine:
     $javacc : java\backslash ABC.jj \rightarrow java\backslash ABCParser.java$
   - Parseri kompileerimine:
     $javac : java\backslash ABCParser.java \rightarrow class\backslash ABCParser.class$
     Koos sellega genereeritakse (ja paigutatakse kataloogi *class*) ka failid
     ABCTokenManager.class,
     ABCTreeConstants.class,
     ABCParserConstants.class,
     JJTABCParserState.class
   - Ülemklassi *BaseLanguage* kompileerimine:
     $javac :$
     $java\backslash BaseLanguage.java \rightarrow class\backslash BaseLanguage.class$
   - Lisatava baaskeele klassi *BaseLanguageABC* kompileerimine:
     $javac :$
     $java\backslash BaseLanguageABC.java \rightarrow class\backslash BaseLanguageABC.class$

Pärast ülalkirjeldatud tegevuste sooritamist ongi süsteemi *AmadeusJSN* lisatud uus baaskeel (*ABC*). Vastavat baaskeele klassi ja uue keele grammatikafaili saab nüüd kasutaja oma vajaduste kohaselt muuta. Esmajoones tuleb muidugi hakata tegelema grammatikafailiga, sest süsteemi poolt on loodud vaid grammatikafaili mall. Baaskeele lisamise protseduuri lõpuks kuvabki süs-

teem uue akna, avades selles grammatikafaili projekti. Viimase all töötades on kasutajal mugav oma keele grammatikat järkjärgult välja arendada.

4.4 Baaskeele automaatne kustutamine

Juhul, kui aktiivsete baaskeelte nimekiri kasvab ülemäära pikaks, on mõistlik mõningad mittevajalikud baaskeeled ära kustutada. Seda arvestades sai süsteemile *Amadeus* lisatud uus funktsioon, mis käivitatakse menüüvalikuga *Base + Remove language*. Selle funktsiooni realiseerimiseks on loodud klass *Item13Listener*, mille struktuur on toodud Lisas 5, ning klassi `BaseLanguage` lisatud uus meetod *userRemoveLang* (samuti Lisas 5).

Baaskeele kustutamiseks peab kasutaja avama uue *Amadeus* akna (*Frame + New*) ja selles uue skeemteksti (*File + New*), seejärel märgistama tekkinud skeemi ja valima *Base + Remove language*. Ilmub momendil aktiivsete baaskeelte loetelu, millest saab kasutaja valida baaskeele, mida ta soovib kustutada.

Nupu *Select* vajutamisel toimub kustutatava baaskeele valideerimine: kontrollitakse, et kasutaja ei kustutaks *Amadeus* töö jaoks vajalikke baaskeeli (*XML*, *Java*, *LaTeX* ja *None*). Juhul kui kasutaja on valinud näiteks *XML*, antakse talle vastav veateade (*Antud nimega baaskeelt kustutada ei ole soovitatav*). Kui kasutaja aga soovib kustutada mõnda muud baaskeelt, kontrollitakse, kas tegemist on aktiivsete baaskeelte nimekirjas oleva viimase baaskeelega. Juhul, kui kustutatav baaskeel on viimane, antakse kasutajale võimalus baaskeele lõplikuks kustutamiseks süsteemist *Amadeus* (*Kas soovite antud baaskeelt lõplikult kustutada?*).
Nupu *YES* vajutamisel kustutataksegi süsteemist *Amadeus* baaskeel lõplikult. Selleks sooritatakse järgmised tegevused (järgnevalt oletame, et kustutava baaskeele nimi on *ABC*):

1. Kustutakse baaskeele projektikirjelduse fail *Portfolio\ABC.html*.

2. Kustutatakse baaskeele grammatikafail *grammars\ABC.jjt* ja kõik selle töötlemise tulemusena loodud failid (süsteemi kataloogides *java* ja *class*).

3. Kustutatakse baaskeele *BaseLanguage* alamklass
   docHtml\BaseLanguageABC.html,
   samuti ka *java\BaseLanguageABC.java*
   ning *class\BaseLanguageABC.class*.

4. Muudetakse *BaseLanguage* klass (nii *BaseLanguage.html* kui ka *BaseLanguage.java*), eemaldades kustutatava baaskeelega seotud parameetrid:

   - Süsteemis kasutatavad baaskeelte konstandite ja nende tähiste loetelu lõpust kustutatakse baaskeelega seotud konstandi kirjeldus
     `final static byte ABC = a;`
   - Aktiivsete baaskeelte nimede loetelust (sõnedemassiivist)
     `static String[ ] items` kustutatakse element `"ABC"`.
   - Dialoogis kuvatavate baaskeelte nimekirja kuuluvate keelte konstantide loetelust (konstantidemassiivist)
     `static byte[ ] currentSetOfBL`
     kustutatakse element `ABC`.
   - Kõikide, nii aktiivsete kui ka passiivsete baaskeelte nimede loetelust (sõnedemassiivist)
     `static String[ ] useditems` kustutatakse element `"ABC"`.
   - Baaskeelte ülemklassist (*BaseLanguage*)
     meetodist *static BaseLanguage newBaseLanguage(byte type)*
     kustutatakse variant
     ```
     case ABC:
     return new BaseLanguageABC();
     ```

5. Kompileeritakse uuesti ülemklass *BaseLanguage.java*.

Nupu *NO* vajutamine[1] jätab aga kasutajale võimaluse tulevikus baaskeel taastada. Antud juhul baaskeele kustutamine süsteemist *Amadeus* ei tähenda täielikku/lõplikku baaskeele eemaldamist süsteemist, vaid tegelikult

[1] Järgnevalt kirjeldatud tehnoloogiat kasutatakse samuti mitteviimase (st. mitte viimasena lisatud) baaskeele kustutamiseks.'

selle nö. passiivseks muutmist. Kõik selle baaskeelega seotud failid tõstetakse ümber baaskeelte repositooriumi (*Personal\BL_repository*) vastavatesse kataloogidesse, millest vajadusel on võimalik seda keelt taastada (vaadeldakse järgmises jaotises). Kustutamise protsessi jooksul sooritatakse järgmised tegevused:

1. Tõstetakse ümber baaskeele projektikirjelduse fail *Portfolio\ABC.html* kataloogi *Personal\BL_repository\ABC\Portfolio*.

2. Tõstetakse ümber baaskeele grammatika fail *grammars\ABC.jjt* kataloogi *Personal\BL_repository\ABC\grammars*.

3. Kustutatakse kõik baaskeele grammatikafaili *grammars\ABC.jjt* töötlemise tulemusena loodud failid (süsteemi kataloogides *java* ja *class*).

4. Tõstetakse ümber baaskeele *BaseLanguage* alamklass *docHtml\BaseLanguageABC.html* kataloogi *Personal\BL_repository\ABC\docHtml*.

5. Muudetakse *BaseLanguage* klass (nii *BaseLanguage.html* kui ka *BaseLanguage.java*), eemaldades kustutatava baaskeelega seotud parameetrid:

   - Süsteemis kasutatavad baaskeelte konstandite ja nende tähiste loetelu lõppust kustutatakse baaskeelega seotud konstandi kirjeldus `final static byte ABC = a;`.
   - Aktiivsete baaskeelte nimede loetelust(sõnedemassiivist) `static String[ ] items` kustutatakse element `"ABC"`.
   - Dialoogis kuvatavate baaskeelte nimekirja kuuluvate keelte konstantide loetelust (konstantidemassiivist) `static byte[ ] currentSetOfBL` kustutatakse element `ABC`.
   - Baaskeelte ülemklassi (*BaseLanguage*) meetodist *static BaseLanguage newBaseLanguage(byte type)* kustutatakse variant

     ```
     case ABC:
     return new BaseLanguageABC();
     ```

6. Kompileeritakse uuesti ülemklass *BaseLanguage*.

Pärast ülalkirjeldatud tegevuste sooritamist ongi süsteemist *AmadeusJSN* lõplikult (või mittelõplikult) kustutatud baaskeel (*ABC*). Kasutajale avatakse süsteemi *Amadeus* üldine projektikirjeldus, kus ta võib tööd jätkata, kuid nüüd juba ilma kustutatud baaskeeleta.

4.5 Baaskeele automaatne taastamine

Süsteemile *Amadeus* on lisatud uus funktsioon, mis käivitatakse menüüvalikuga *Base + Restore language*. Selle funktsionaalsuse realiseerimiseks on loodud klass *Item14Listener*, mille struktuur on toodud Lisas 7. Baaskeele taastamise protsess on sarnane baaskeele lisamise protsessiga. Vahe on selles, et vajalikud failid on juba olemas baaskeelte reporsitooriumis (*Amadeus* süsteemi *Personal\BL_ repository* kataloogis) ja need lihtsalt tõstetakse ümber vastavatesse kataloogidesse.

Baaskeele taastamiseks peab kasutaja avama uue *Amadeus* akna (*Frame + New*) ja selles uue skeemteksti (*File + New*), seejärel märgistama tekkinud skeemi ja valima *Base + Restore language*. Ilmub aktiivsete ja passiivsete baaskeelte loetelu, millest saab kasutaja valida taastatava baaskeele,

Peale nupu *Select* vajutamist toimub baaskeele nime valideerimine:

1. kui valitud baaskeel on juba aktiivne ja kasutusel, antakse kasutajale hoiatus (*Antud nimega baaskeel on juba aktiivne*);

2. kui baaskeel eksisteeris, kuid oli eemaldatud enne käesoleva süsteemi rakendamist, antakse kasutajale hoiatus (*Antud nimega baaskeelt taastada ei ole võimalik*).

Pärast baaskeele nime (edukat) valideerimist toimub taastatava baaskeele automaatne lisamine süsteemi *AmadeusJSN*, mille käigus sooritatakse järgmised tegevused (järgnevalt eeldame, et taastame baaskeelt *ABC*):

1. Projektikirjelduse faili kopeerimine baaskeelte repositooriumist
   Personal\BL_ repository\ABC\Portfolio\ABC.html
   kataloogi *Portfolio\ABC.html*

2. Grammatikafaili *ABC.jjt* kopeerimine baaskeelte repositooriumist
   Personal\BL_repository\ABC\grammars\ABC.jjt
   kataloogi (*grammars\ABC.html*)

3. *BaseLanguage* klassi alamklassi *BaseLanguageABC* kopeerimine baaskeelte repositooriumist
   Personal\BL_repository\ABC\docHtml\BaseLanguageABC.html
   kataloogi *docHtml\BaseLanguageABC.html*.

4. Failis *BaseLanguage.html* taastatava baaskeelega seotud parameetrite muutmine (täpselt samuti, nagu uue baaskeele lisamisel, vt. jaotis 4.3).

5. Failide automaatne töötlemine. Järgnevas tegevuste loetelus on eeldatud, et lisatava baaskeele nimeks on *ABC*. Failinimede ette on veel lisatud vastava süsteemi kataloogi nimetus.

   - Grammatikafaili teisendamine:
     $jjtree : grammar\backslash ABC.jjt \rightarrow java\backslash ABC.jj$
     Protsessori *jjtree* poolt genereeritakse (ja paigutatakse kataloogi *java*) ka failid
     ABCTokenManager.java,
     ABCTreeConstants.java,
     ABCParserConstants.java,
     JJTABCParserState.java
   - Parseri lähtefaili genereerimine:
     $javacc : java\backslash ABC.jj \rightarrow java\backslash ABCParser.java$
   - Parseri kompileerimine:
     $javac : java\backslash ABCParser.java \rightarrow class\backslash ABCParser.class$
     Koos sellega genereeritakse (ja paigutatakse kataloogi *class*) ka failid
     ABCTokenManager.class,
     ABCTreeConstants.class,
     ABCParserConstants.class,
     JJTABCParserState.class

- Ülemklassi *BaseLanguage* kompileerimine:
  $javac$:
  $java\backslash BaseLanguage.java \rightarrow class\backslash BaseLanguage.class$
- Lisatava baaskeele klassi $BaseLanguageABC$ kompileerimine:
  $javac$:
  $java\backslash BaseLanguageABC.java \rightarrow class\backslash BaseLanguageABC.class$

Lõpuks avatakse kasutajale taastatud baaskeele projektikirjeldus, kus võib ta jätkata tööd taastatud baaskeelega.

4.6 Baaskeele grammatika käitlemise tugi

Käesolevas jaotises spetsifitseeritakse baaskeele projekt, mis on mõeldud eeskätt grammatikafaili töötlemiseks.

Jaotises 3.4 oli kirjeldatud uue baaskeele lisamise protsess. Viimane, aga ka sellele järgnev grammatikafaili iteratiivse muutmise protsess, on selgelt väga töömahukad. Kasutajal tuleb grammatikafaili ja baaskeele klassi muutmisel töötada paljude failidega, mida peale iga muudatust on vaja salvestada ja kompileerida/töödelda.

Probleemi lahendamiseks sai otsustatud, et kohe juba baaskeele lisamisel on mõistlik luua baaskeele projekt, kus kasutajal oleks mugav ja lihtne grammatikafailiga ning baaskeele klassiga töötada. Baaskeele projektikirjelduse struktuur (joonis 4.6) on standardne ja sisaldab järgmist:

1. projekti nimi, tavaliselt baaskeele nimi;
2. *Path*, *Save-script* ja *Open-script* rühma; neis olevad atribuutide süsteemsed kirjeldused võetakse *Amadeus.properties* failist;
3. projekti jaoks vajalike failide haru, mis jaguneb omakorda kolmeks:
   - Failid *BaseLanguage.html* ja *BaseLanguageKEEL.html*.
   - Baaskeele grammatikafaili ja parseri *Path*, *Save-script* ja *Open-script* rühmade atribuutide kirjeldused ning failide nimed. Kõige olulisem on *KEEL.jjt*, mida tuleb kasutajal muuta/täiendada vajaduste kohaselt.

- Projekti failid – projekti KEEL all kasutaja poolt loodavad näitefailid.

Nagu eelnevalt öeldud, avatakse kasutajale (uue baaskeele automaatsel lisamisel) uue baaskeele projektikirjeldus, kus võib ta oma tööd (peamiselt grammatikafailiga) jätkata.

Meenutame, et gammatikafaili muutmisega kaasneb kolme faili töötlemine:

- baaskeele grammatikafaili *KEEL.jjt* teisendamine failiks *KEEL.jj*;
- parseri lähtefaili genereerimine: *KEEL.jj* → *KEELParser.java*;
- parseri kompileerimine: *KEELParser.java* → *KEELParser.class*.

Selleks peaks kasutaja failid eraldi avama ja kompileerima toodud järjekorras: esialgu *KEEL.jjt*, siis *KEEL.jj* ja lõpuks *KEELParser.java*.

Võib oletada, et selline grammatikafaili muutmise protsess osutub väga tülikaks. Probleemi lahendab projektikirjelduse sobivalt defineeritud atribuutide osa. Nimelt on baaskeele projektikirjelduse faili alamprojekti *Grammatikad* (joonis 4.7) *Path* ja *Save-script* rühmad konstrueeritud järgmiselt:

- *Path-script* atribuutides leiduvad kaks parameetrit, millega defineeritakse parseri lähtefaili (*JAVACC*) ja selle (*JAVAPARSER*) kompileerimiste eeskirjad
- *Save-script Process* atribuut oli täiendatud eelmises punktis kirjeldatud *JAVACC* ja *JAVAPARSER* parameetritega

Tänu sellistele atribuutidele toimubki *KEEL.jjt* faili salvestamisel iga kord eelpool kirjeldatud failide automaatne kompileerimine:
KEEL. jjt –> KEEL.jj –> KEELParser.java –> KEELParser.class.

Kokkuvõttes, grammatikafaili iteratiivse muutmise tülikuse probleem on lahendatud baaskeele projektikirjelduse loomisega ning selle täiendamisega lisaatribuutidega, et võimaldada grammatikafaili automaatset kompileerimist.

4.7 Süsteemi *AmadeusJSN* testimine

Süsteemi *AmadeusJSN* esialgse testimise aluseks oli töö kahe "mängu-keelega". Lähtekeeltena kasutati eespool kirjeldatud metsa vasaku suluesituse keelt (*VSE*) ja analoogilst, metsa parema suluesituse keelt *PSE*. Katsetati vastavate baaskeelte lisamist, kustutamist, taastamist. Mõlema keele korral realiseeriti autori poolt nii skeemistid (grammatika-põhiselt) kui ka tekstualiseerijad ("käsitsi").

Reaalseks praktiliseks (juhendaja poolt teostatud) testjuhuks on programmeerimiskeele *Java vers.*1.5 edukas integreerimine süsteemi *AmadeusJSN*. Vastav skeemistusmudel leidub Lisas 9. Integreerimise tulemusena saab nüüd süsteemi nt. kasutada selle programmeerimiskeele lähtekoodi skeemkujul vaatlemiseks [3].

Käesolevas jaotises vaatleme mõningaid esialgse testimisega (keeltega *VSE* ja *PSE*) seotud küsimusi.

4.7.1 Baaskeel *PSE* lisamine

Metsa parema suluesituse keel *PSE* on väga sarnane keelega *VSE*. Alljärgnevas selgitame selle keele automaatse lisamise käiku üksikute sammude kaupa.

1. Keele *PSE BNF* grammatika:
   ```
   metsaPSE::="("puuPSE ("," puuPSE)* ")"
   puuPSE::= [metsaPSE] juur
   juur::=IDENTIFIER
   ```
   Märgime, et selle kohaselt näeks töö kolmandas peatükis (joonisel 3.1) näitena toodud mets esitub kirjena `((B,(D,E)C,F)A,(Y)X)`. Erinevalt vasakust suluesitusest paikneb puu juur oma alampuude metsa järel.

2. Süsteemi lisatakse baaskeel nimega *PSE*, kasutades *AmadeusJSN* automaatseid vahendeid (*Base + Add new language*). Selle tulemusena luuakse nii antud baaskeelega seotud projekt kui ka kõik muud seonduvad failid.

3. Töötades projekti all arendatakse *PSE* keele grammatikafail *BNF* grammatikast lähtudes. Antud juhul ei kasutatud grammatika skeemkuju, vaid *JJTree*-grammatika koostati lihtteksti failina (Lisa 10).

4. Grammatikafaili töötlemisel (projekti all toimetetava grammatikafaili salvestamisel, *File + Save*) luuakse automaatselt *PSE* baaskeele parser.

5. Lähtudes konkreetse näitemetsa (redutseeritud) parsipuust (joonis 4.8) koostatakse keele *PSE* skeemkonversiooni mudel (joonisel 4.9). Nagu näha, osutub see täiesti analoogiliseks varem tutvustatud mudeliga keele *VSE* jaoks.

6. Programmeeriti klassi *BaseLanguagePSE* meetod *textualize*. Selle meetodi kood on päris lihtne ja lühike (vt. Lisa 11).

Sel teel, keele *PSE* eduka integreerimisega, sai veelkord testitud uue baaskeele automaatse lisamise funktsioon uues süsteemis.

4.7.2 Keelte *VSE* ja *PSE* vastastikune konverteerimine

Pärast seda, kui mõlema uue baaskeele jaoks olid realiseeritud nii skeemistamine kui ka tekstualiseerimine, sai testimise eemärgil katseatud uut süsteemi kui konverterit tüüpi $PSE \rightarrow VSE$. Konverteerimise protsess on kujutatud joonisel 4.10.

Oletame, et on olemas lähtekeele *PSE* lihttekst, nt `((E,(D,C)B)A,(Y)X)`, mida soovitatakse konverteerida sihtkeele *VSE* lihttekstiks. Antud näite korral oleks siis oodatavaks korrektseks tulemuseks `(A(E,B(C,D)),X(Y))`.

Konverteerimine süsteemi *AmadeusJSN* abil on väga kerge. Esiteks tuleb avada uus aken (*Frame + New*) ja selles kujutada konverteeritav lähtetekst. Viimase võib otse tippida klaviatuurilt uuena avatud (*File + New*) tühja skeemteksti või siis sisestada olemasolevast lihtteksti failist (*File + Read text*). Konverteeritava teksti baaskeeleks tuleks omistada *PSE* (*Base + Set ... PSE*). Seejärel sooritatakse skeemkonversioon $PSE \rightarrow SK$ (*Tools + Sketchify*), uues aknas kuvatakse lähtetekstis esitatud mets hierarhilisel skeemkujul. Kui viimasele omistada nüüd baaskeeleks *VSE* (*Base + Set ...*

VSE) ja rakendada tekstualiseerimist (*Tools + Textualize*), saadaksegi uues aknas selle sama metsa vasak suluesitus.

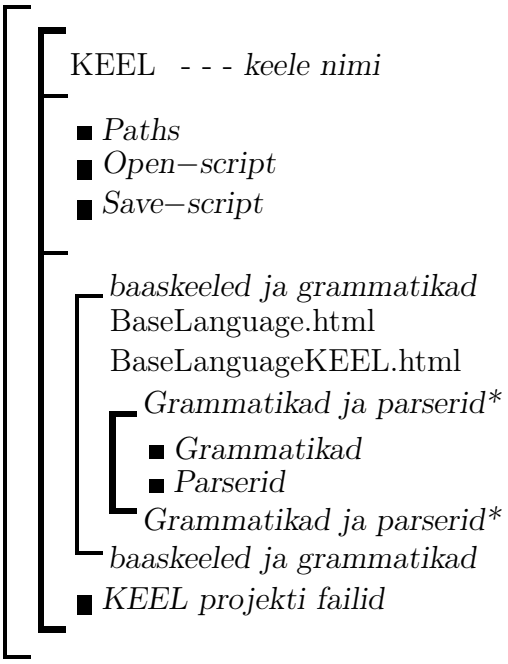
Tulemus lähteteksti `((E,(D,C)B)A,(Y)X)` korral on kujutatud joonisel 4.11. Vertikaalne kuju on määratud meetodiga *textualize* klassis *BaseLanguageVSE*, mis lisab tulemusse reavahetusi ja eeltühikuid metsa struktuuri paremaks väljatoomiseks. Meenutame, et grammatika kohaselt tühemikke (nagu reavahetuse ja tühiku sümbolid) ignoreeritakse.

Süsteemi on testimise eesmärgil katsetatud ka konverterina tüüpi $VSE \rightarrow PSE$. Sellesuunalise konverteerimise käik on ülalkirjeldatuga täisesti sümmeetriline.

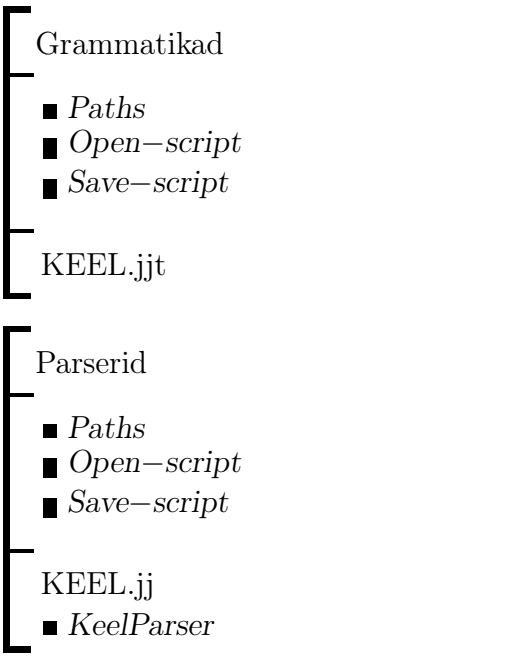
4.8 Kokkuvõtteks

Töö põhitulemuseks on süsteemi *Amadeus* uus versioon *AmadeusJSN*, millele on lisatud automatiseerimise tehnilised lahendused baaskeeltega manipuleerimiseks: uue baaskeelte lisamine, olemasoleva kustutamine ja kustutatud baaskeele taastamine. Samuti on kirjeldatud süsteemi kataloogi *Amadeus* üldine ülesehitus ja antud ülevaade *Amadeus.properties* failist, kust võetakse erinevad parameetrid süsteemi *Amadeus* korrektseks töötamiseks. Lugejale tutvustatakse süsteemi *Amadeus* projekti mõiste. Grammatikafaili iteratiivse arendamisetoeks on spetsifitseeritud vastav baaskeele projektikirjeldus. Töö üks eesmärkidest oli muuta kasutaja töö mugavamaks ja lihtsamaks baaskeeletega manipuleerimisel süsteemis *Amadeus*.

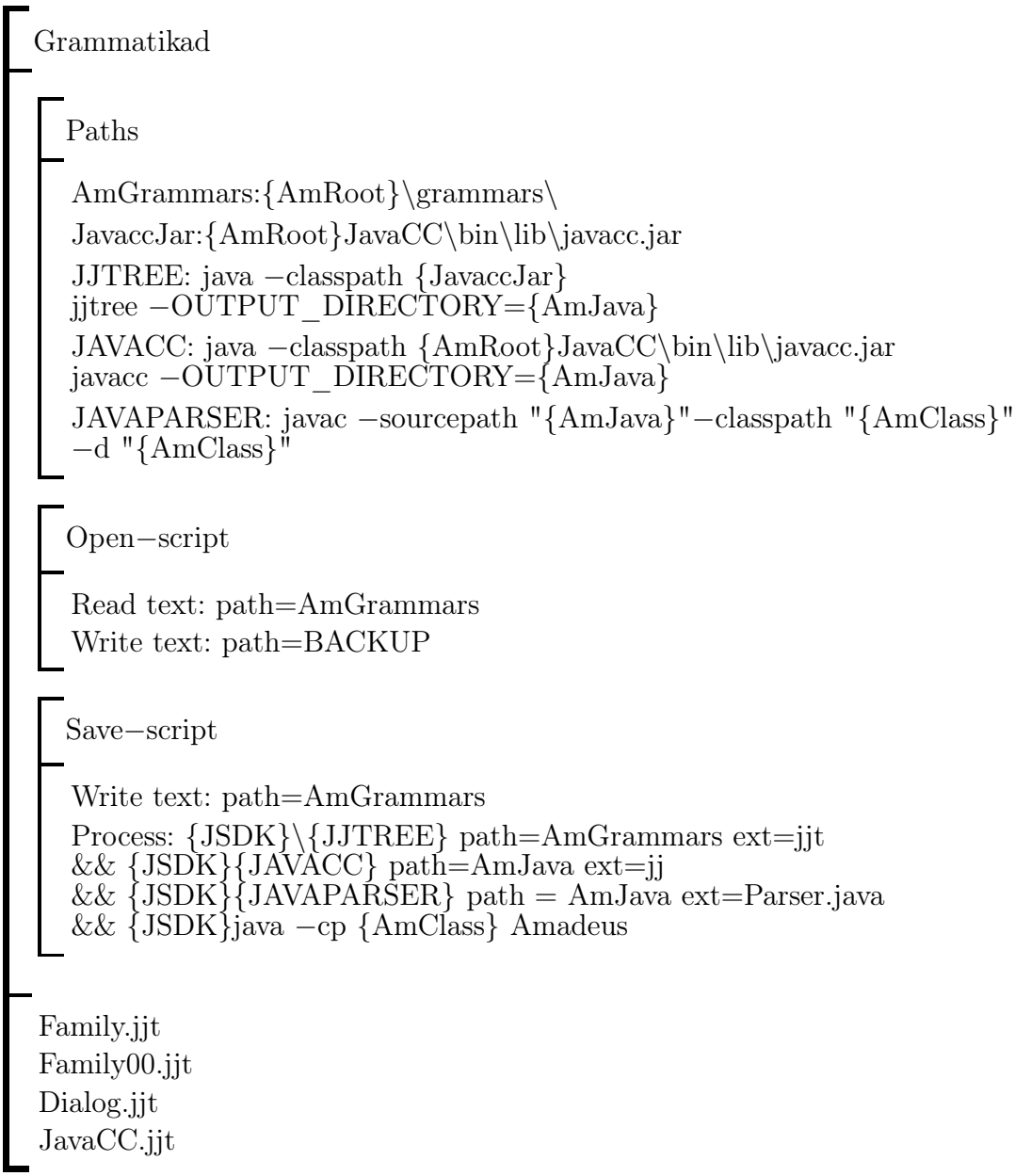
Loodud versioon on testitud lihtsate lähtekeelte (metsa vasaku/parema suluesituse keelte) integreerimise teel. Samuti on süsteemi juba reaalselt rakendatud programmeerimiskeele *Java* (*vers.*1.5) skeemisti loomiseks.



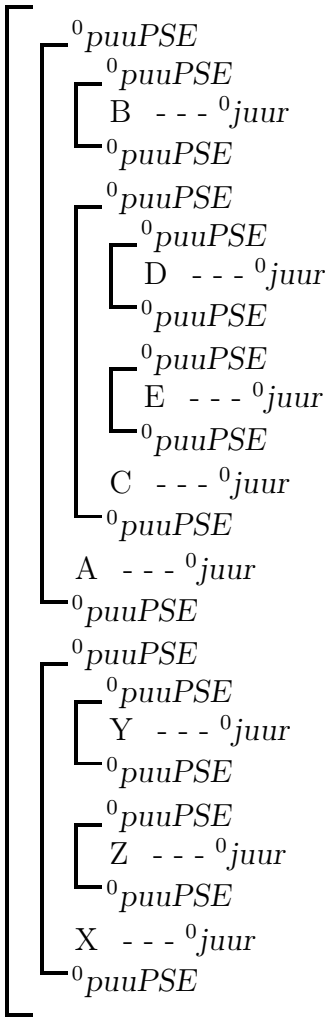
Grammatikad ja parserid



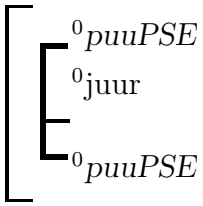
Joonis 4.6: Baaskeele projektikirjeldus



Joonis 4.7: Alamprojekti *Grammatikad* atribuudirühmad *Path-script* ja *Save-script*.

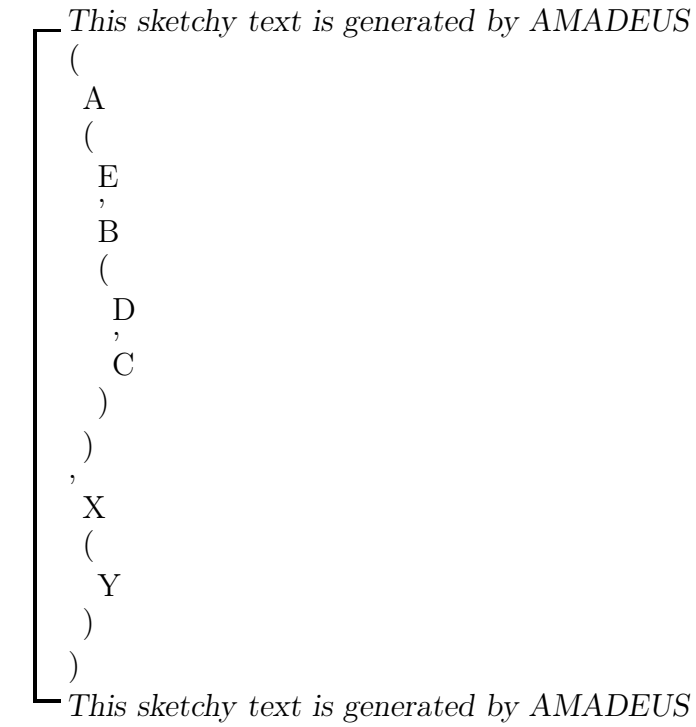


Joonis 4.8: *PSE* baaskeele näiteteksti `((B,(D,E)C)A,(Y,Z)X)` redutseeritud parsipuu



Joonis 4.9: Baaskeele *PSE* skeemmudel

Joonis 4.10: *PSE*-keelse teksti konverteerimine *VSE*-keelseks tekstiks.



Joonis 4.11: *PSE*-keelse teksti konverteerimistulemus *VSE*-keelseks tekstiks.

Kokkuvõte

Aktuaalseks küsimuseks arvutikeelse info pideva suurenemise tõttu on eriotstarbeliste arvutikeelte teisendajate, sh. konverterite loomine. Enamasti on aga eriteisendaja loomine väga kulukas. Töös keskendutakse just konverteriarendamise probleemile.

Üha suurenev vajadus erinevat tüüpi konverterite järele on tõstatanud loomuliku küsimuse, kuidas oleks võimalik automatiseerida konverterite loomist. Muidugi on soovitav luua võimalikult üldist tüüpi konvertereid. Kõige ideaalsem oleks üks universaalne konverter tüüpi *Any-to-Any*. Nii üldisel kujul püstitatud ülesanne oleks ilmselt lahendamatu, kuid peale probleemi mõistlikku kitsendamist saab teha olulisi reaalseid samme konverteriarenduse automatiseerimiseks. Probleemi võib sõnastada järgmiselt: on antud lähtekeel, sihtkeel ja konverteerimise mudel, ülesandeks on automaatselt genereerida vastav konverter. Töös selgitatakse konverteriarenduse võimalusi süsteemi *Amadeus* platvormil.

Käesolevas töös välja töötatud lähenemisviis võib näida esmapilgul küll piiratud, eeskätt sihtkeele (milleks on skeemkeel) fikseeritust arvestades, kuid samas iseloomustab seda universaalsus lähtekeelte pere osas: ainasaks nõudeks lähtekeelele on selle formaalse grammatika olemasolu. Pealegi on väljatöötatud süsteemis võimalik arendada ka üldist tüüpi konvertereid, kasutades skeemkeelt konverteermise vahekeele rollis.

Antud väitekirja põhitulemuseks on süsteemi *Amadeus* uus versioon *AmadeusJSN*, mis on saadud lähteversioonist *AmadeusJS* konverteriloomet toetavate erisuste lisamise teel, ning kus on võimalik skeemkonverterite kiire arendamine. Põhiliseks võtmeküsimuseks on sealjuures uue baaskeele integreerimine süsteemi ning baaskeeltega töötamine ja grammatikafaili re-

digeerimine, samuti skeemistamise ja tekstualiseerimise meetodite efektiivne realiseerimine.

Skeemkonverteri arendamiseks süsteemis *Amadeus* tuleb spetsifitseerida lähtekeele grammatika ja integreerida vastav uus baaskeel. Viimases tuleb realiseerida vähemalt parsimise ning skeemistamise osad. Skeemkonverteri kasutamiseks tuleb eelnevalt veel konkretiseerida konversiooni mudel. Taolist moodust vahetult kasutades on koostöös juhendaga (J. Kiho) loodud konverter *JJTree/JavaCC* kujul kirjutatud lihttekstiliste grammatikafailide viimiseks skeemkujule. Vastavalt lisati uus ka baaskeel nimega *JavaCC*.

Automatiseeritud on baaskeeltega manipuleerimine (uue baaskeele lisamine, olemasoleva kustutamine, kustutatud baaskeele taastamine). Baaskeele grammatikafaili iteratiivse arendamise toeks on spetsifitseeritud vastav projektikirjeldus, millega töötades on kasutaja töö mugav ja efektiivne.

Loodud versioon on testitud lihtsate lähtekeelte (metsa vasaku/parema suluesituse keelte) integreerimise teel. Samuti on süsteemi juba reaalselt rakendatud programmeerimiskeele *Java* (*vers.*1.5) skeemisti loomiseks.

Grammar-based software converter development

Master work

Natalia Tomassova

Abstract

The software conversion process is understood as a lightweight translation of computer texts, focused on changing the outer form of texts. Currently, one of the most popular conversion types is $Some \rightarrow XML$, where texts in some data or program description language are re-formatted into certain *XML* format.

The aim of this thesis is to investigate possibilities of automation of converter development processes.

Overviews of some examples of existing converters (*DELL, XFlat, Link-Grammar*) are presented. The notion of software converter is refined. The main features of the existing multi-editor *AmadeusJS* are introduced. It uses a special hierarchical (sketchy) form for texts under processing. Therefore, the conversion tasks ($Some \rightarrow SketchyText$, and *vice verse*) are inherent for *AmadeusJS*, and it can naturally be considered as a framework for converter development.

A language, incorporated into *Amadeus* is called the base language. For any base language LK, the converters of types $LK \rightarrow SketchyText$ and $SketchyText \rightarrow LK$ are implemented. So, to get a converter $LK' \rightarrow LK''$, the both languages, LK' and LK'' are to be incorporated. After that, the sys-

tem is ready to perform needed conversion by sequentially applying $LK' \rightarrow SketchyText$ and $SketchyText \rightarrow LK''$, provided that necessary conversion model is defined.

The thesis results in creating the new version of *Amadeus* system – *AmadeusJSN*. It enables to incorporate automatically new base languages: the user has only to define the grammar in the *JJTree* form. Also, it supports iterative developing and debugging of new grammars. For that purpose, the project facility of *Amadeus* is applied. Additional base language operations, such as new base language addition, base language removing and restoration of removed early base language, are also implemented in *AmadeusJSN*.

Kirjandus

[1] J.Kiho. *SKM. Sketchy Modeling of Computer Texts*. 18th January 2000, 64 p.

[2] J.Kiho, R.Ustitch, M.Tudre. *Designing Computer Texts with AMADEUS. In: J.Penjam (Ed.) Software Technology. Proceedings of the Fenno-Ugric Symposium FUSST'99. Technical Report CS 104/99. Institute of Cybernetics at Tallinn Technical University*. Tallinn 1999, 151-162.

[3] J.Kiho, *Amadeus-Jasovi-5*.
`http://www.cs.ut.ee/~kiho/Amadeus-Jasovi-5/`

[4] K.Heero, *Parsigeneraator JavaCC ja skeemmodelleerimine*. Magistritöö. TÜ, 2002, 93 lk.

[5] J.Kiho. *Algoritmid ja andmestruktuurid*. Kolmas, parandatud ja täiendatud trükk. TÜ, 2003, 147 lk.

[6] A.Reitsakas, *Analüüsivahend JJTree/JavaCC ja selle rakendamine toimetiarenduses*. Magistritöö. TÜ, 2003, 69 lk.

[7] *JavaCC: Grammar Files*.
`https://javacc.dev.java.net/doc/javaccgrm.html`

[8] *JavaCC[TM]: JJTree Reference Documentation*.
`https://javacc.dev.java.net/doc/JJTree.html`

[9] *Overview of XML Convert and XFlat*.
`http://www.unidex.com/overview.htm`

[10] *W3C, DEL-Data Extraction Language.* `http://www.w3.org/TR/data-extraction`

[11] *X-FETCH PERFORMER[TM]. Fast Application Development in Java and XML Environments.* `http://www.x-fetch.com/performer.html`

[12] *Link Grammar.* `http://www.link.cs.cmu.edu/link/`

[13] J.Kiho, S.Solopova. *Heterogeneous File Projects in the Sketchy Modeling Environment. The Tenth Nordic Workshop on Programming and Software Development Tools and Techniques.* IT-University, Copenhagen, August 18-20, 2002, pp. 71-79

[14] S.Golovko, *Projekti mõiste ja teostus süsteemis Amadeus.* Magistritöö, TÜ, 2004, 72 lk.

[15] *Extensible Markup Language (XML)*, `http://www.riik.ee/xml/trans/REC-xml-19980210-ee.html`

Indeks

Lisad

Lisa 1. Keele *VSE JJTree*-grammatika lihtteksti kujul
Lisa 2. Baaskeele klassi mall
Lisa 3. Ülemklass *BaseLanguage*
Lisa 4. Klassi *Item12Listener* struktuur
Lisa 5. Klass *Item13Listener* ja meetod *userRemoveLang*
Lisa 6. Klassi *Item14Listener* struktuur
Lisa 7. JavaCC grammatikafail
Lisa 8. *JavaCC* skeemmudel
Lisa 9. *Java5* skeemmudel
Lisa 10. Keele *PSE JJTree*-grammatika lihtteksti kujul
Lisa 11. Klass *BaseLanguagePSE*
Lisa 12. *AmadeusJSN* ja käesoleva magistritöö e-koopia

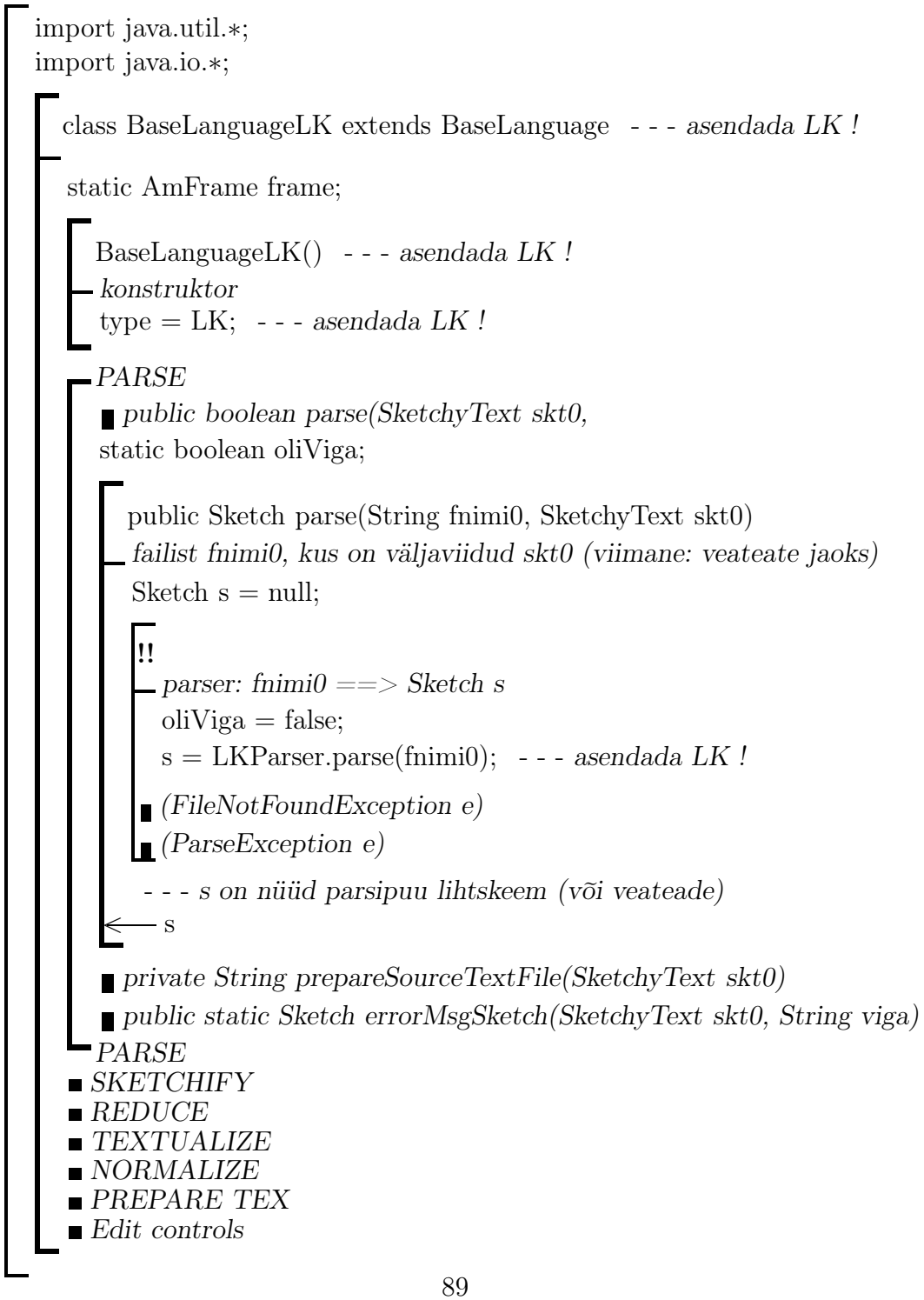
Lisa 1.
Keele *VSE JJTree*-grammatika lihtteksti kujul

```
options { JAVA_UNICODE_ESCAPE = true; STATIC = false; }

PARSER_BEGIN(VSEParser)

import java.io.*;

public class VSEParser {
static Token t;
static Sketch parse(String inputFileName) throws ParseException,
FileNotFoundException {
VSEParser parser = new VSEParser(new FileReader(inputFileName));
SimpleNode t = parser.Start();
Sketch parsipuu = BaseLanguageParseTree.toSketch
(t, VSEParserTreeConstants.jjtNodename);
return(parsipuu);
}

PARSER_END(VSEParser)

SPECIAL_TOKEN : {
" "
| "\t"
```

```
| "\r"
| "\f"
| "\n"
}

/* IDENTIFIERS */
TOKEN :
{
< IDENTIFIER: <LETTER> (<LETTER> | <DIGIT>)* >
| < # LETTER: [ "A-"Z","_", "a"- "z"] >
| < # DIGIT: [ "0"- "9"] >
}

/* SEPARATORS */
TOKEN : {
< LPAREN: "(" >
| < RPAREN: ")" >
| < COMMA: "," >
}

TOKEN :
{
< ANYTHING_ELSE: ( ~ [ ] ) >
}

/*productions-produktsioonid*/
void tIdentifier(): {}{
t = <IDENTIFIER>
{jjtThis.addText(t.image);}}

/** THE GRAMMAR SPECIFICATION STARTS HERE **/
SimpleNode Start() : {} // algussümbol
{
metsaVSE()
```

```
<EOF>
{ return jjtThis; }
}
Token metsaVSE() #void:
{
Token t = getToken(1);
}
{
"("puuVSE() (","puuVSE())* ")"
{ return t;}
}
void puuVSE():
{}
{
juur() [metsaVSE()]
}
void juur() :
{}
{
(tIdentifier())
}
```

Lisa 2.
Baaskeele klassi mall


import java.util.∗;
import java.io.∗;
class BaseLanguageLK extends BaseLanguage - - - asendada LK !
static AmFrame frame;
BaseLanguageLK() - - - asendada LK !
konstruktor
type = LK; - - - asendada LK !
PARSE
public boolean parse(SketchyText skt0,
static boolean oliViga;
public Sketch parse(String fnimi0, SketchyText skt0)
failist fnimi0, kus on väljaviidud skt0 (viimane: veateate jaoks)
Sketch s = null;
!!
parser: fnimi0 ==> Sketch s
oliViga = false;
s = LKParser.parse(fnimi0); - - - asendada LK !
(FileNotFoundException e)
(ParseException e)
- - - s on nüüd parsipuu lihtskeem (või veateade)
s
private String prepareSourceTextFile(SketchyText skt0)
public static Sketch errorMsgSketch(SketchyText skt0, String viga)
PARSE
SKETCHIFY
REDUCE
TEXTUALIZE
NORMALIZE
PREPARE TEX
Edit controls

Lisa 3.
Ülemklass *BaseLanguage*

import java.awt.∗;
import java.awt.event.∗;
class BaseLanguage
abstract class BaseLanguage - - - *baaskeelte ülemklass*

LIIDES
- - - *väljastavad ainult veateate:*
- *public boolean sketchify(SketchyText skt0,*
- *public boolean textualize(SketchyText skt0,*
- *public boolean normalize(SketchyText skt0,*
- *public boolean parse(SketchyText skt0,*
- *void reduce(Sketch s)*
- *void prepareTex(Sketch s)*
- - - *valmis meetodid, võib ka üledefineerida:*
- *byte nextPrimitiveMemberType(byte prev)*
- *byte nextSchemeType(Scheme s)*
- *byte typeForBranchHead(Sketch s)*
- *byte nextPrimitiveHeadType(Scheme s, byte prev)*

LIIDES
*MUUDETAVAD**
- *KEELTE KONSTANDID*
- *KEELTE NIMED*
- *KASUTATAVATE KEELTE LOETELU*
- *KÕIGI KASUTATUD KEELTE NIMED*
- *newBaseLanguage*

*MUUDETAVAD**
- *KLASSIVÄLJAD JA -MEETODID*

class BaseLanguage

——————————————**MUUDETAVAD**

KEELTE KONSTANDID
⋆ static final byte

JAVA = 0;
LATEX = 1;
. . .
VSE = 13;
- - - *uus konstant lisatakse loetelu lõppu*
KEELTE KONSTANDID

KEELTE NIMED
static String[] items =

"Java"
,"LaTex"
. . .
,"VSE"
- - - *uus nimi lisatakse loetelu lõppu*

;
KEELTE NIMED

KASUTATAVATE KEELTE LOETELU
kasutatavate keelte konstandid (suvalises järjekorras)
static final byte[] currentSetOfBL =

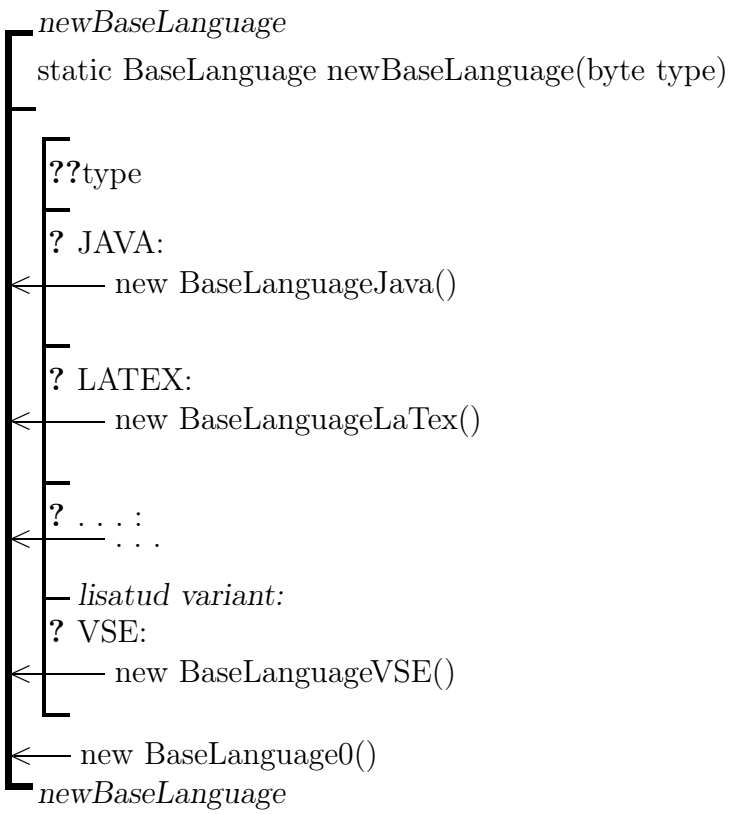
JAVA
,LATEX
. . .
,VSE

;
KASUTATAVATE KEELTE LOETELU

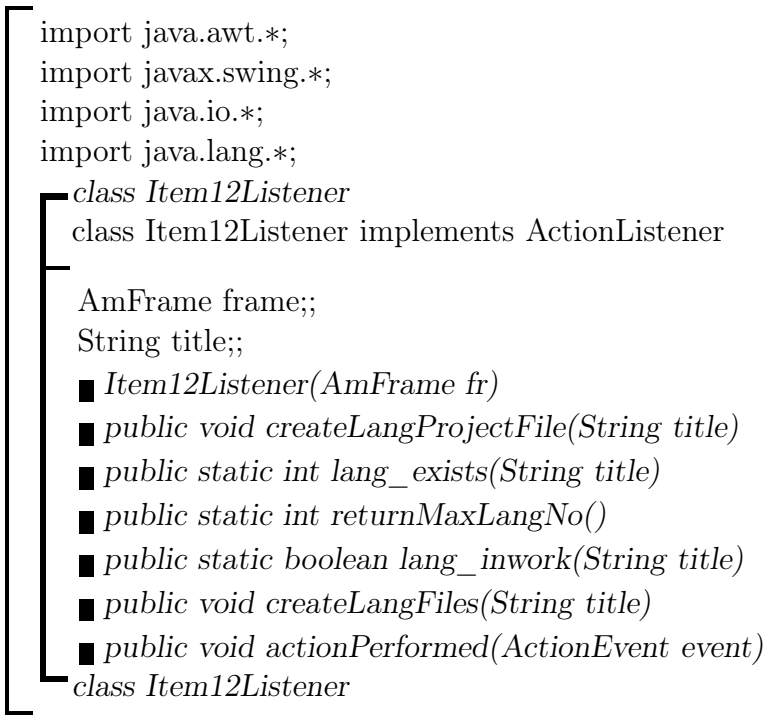
KÕIGI KASUTATUD KEELTE NIMED
static String[] useditems =

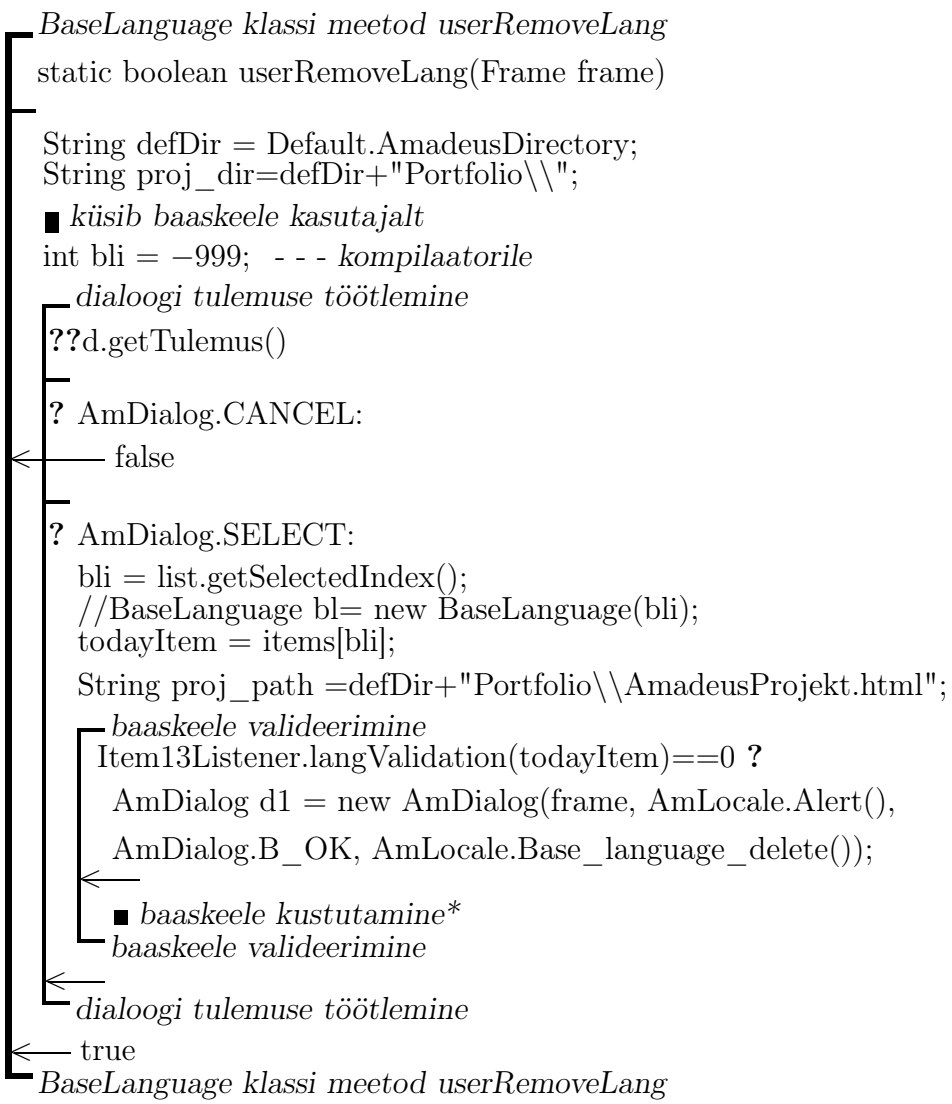
"Java"
,"LaTex"
. . .
,"VSE"

;
KÕIGI KASUTATUD KEELTE NIMED


newBaseLanguage
static BaseLanguage newBaseLanguage(byte type)
??type
? JAVA:
new BaseLanguageJava()
? LATEX:
new BaseLanguageLaTex()
? . . . :
. . .
lisatud variant:
? VSE:
new BaseLanguageVSE()
new BaseLanguage0()
newBaseLanguage

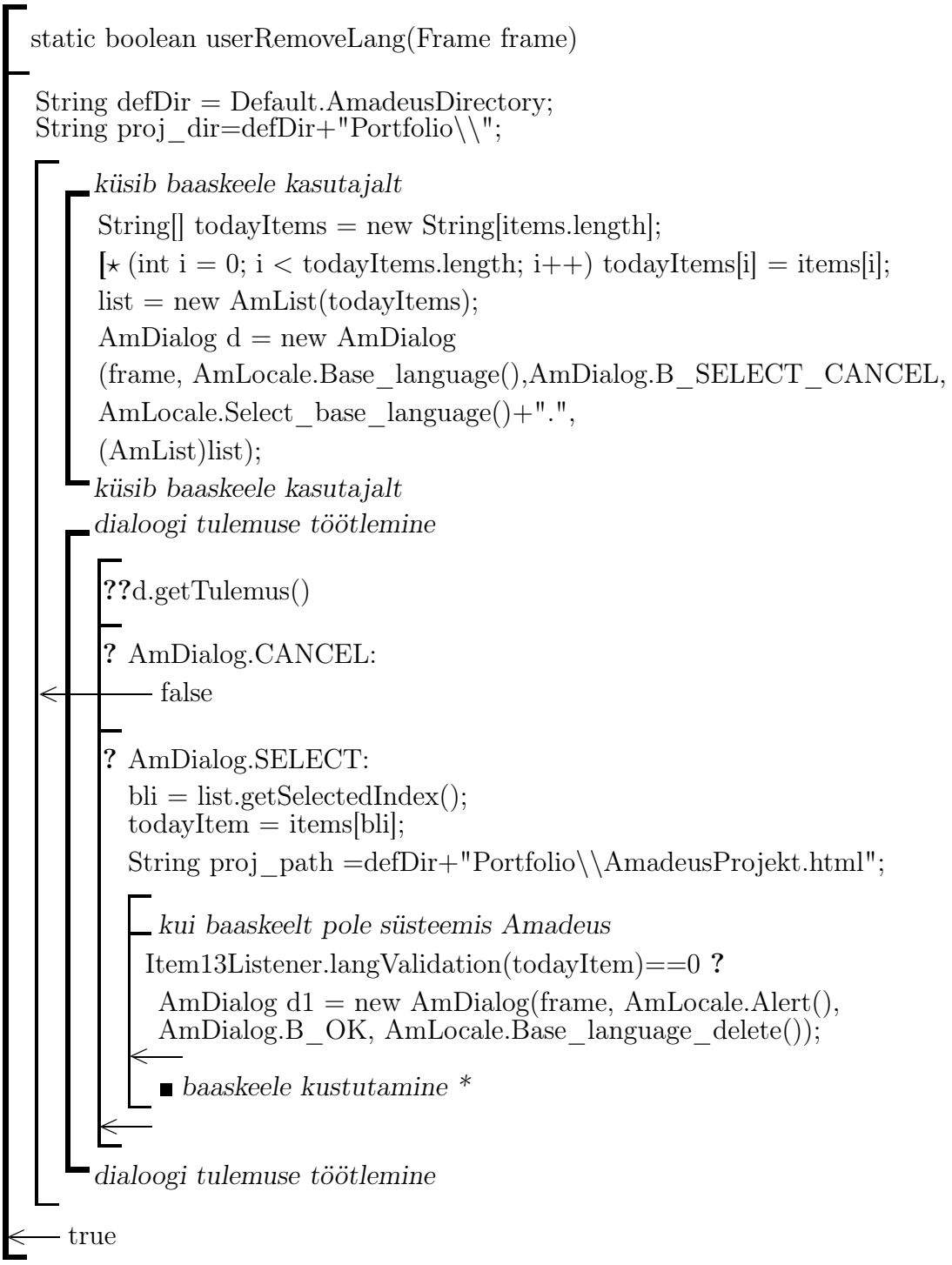
Lisa 4.
Klassi *Item12Listener* struktuur




BaseLanguage klassi meetod userRemoveLang
static boolean userRemoveLang(Frame frame)
String defDir = Default.AmadeusDirectory;
String proj_dir=defDir+"Portfolio\\";
küsib baaskeele kasutajalt
int bli = −999; - - - kompilaatorile
dialoogi tulemuse töötlemine
??d.getTulemus()
? AmDialog.CANCEL:
false
? AmDialog.SELECT:
bli = list.getSelectedIndex();
//BaseLanguage bl= new BaseLanguage(bli);
todayItem = items[bli];
String proj_path =defDir+"Portfolio\\AmadeusProjekt.html";
baaskeele valideerimine
Item13Listener.langValidation(todayItem)==0 ?
AmDialog d1 = new AmDialog(frame, AmLocale.Alert(),
AmDialog.B_OK, AmLocale.Base_language_delete());
baaskeele kustutamine*
baaskeele valideerimine
dialoogi tulemuse töötlemine
true
BaseLanguage klassi meetod userRemoveLang

Lisa 5. Klass *Item13Listener* ja meetod *userRemoveLang*

```
import java.awt.*;
import javax.swing.*;
import java.io.*;
import java.lang.*;

class Item13Listener implements ActionListener

static AmFrame frame;;
String title;;
 Item13Listener(AmFrame fr)
 public static void removeLangProjectFile(String title)
 public static int langValidation(String title)
 public static void backupDir(String path,String title)
 public static void backupDirCreate(String path)
 public static boolean deleteDirectory(File path)
 public static void copyFile(String kust, String kuhu)
 public static void deleteFile(String kust)
 public static void openProjectFile(String proj_path)
 public static void removeLangFiles(String title)
 public static void removeLangFilesFinal(String title)
 public void actionPerformed(ActionEvent event)
```


static boolean userRemoveLang(Frame frame)
String defDir = Default.AmadeusDirectory;
String proj_dir=defDir+"Portfolio\\";
küsib baaskeele kasutajalt
String[] todayItems = new String[items.length];
⋆ (int i = 0; i < todayItems.length; i++) todayItems[i] = items[i];
list = new AmList(todayItems);
AmDialog d = new AmDialog
(frame, AmLocale.Base_language(),AmDialog.B_SELECT_CANCEL,
AmLocale.Select_base_language()+".",
(AmList)list);
küsib baaskeele kasutajalt
dialoogi tulemuse töötlemine
??d.getTulemus()
? AmDialog.CANCEL:
false
? AmDialog.SELECT:
bli = list.getSelectedIndex();
todayItem = items[bli];
String proj_path =defDir+"Portfolio\\AmadeusProjekt.html";
kui baaskeelt pole süsteemis Amadeus
Item13Listener.langValidation(todayItem)==0 ?
AmDialog d1 = new AmDialog(frame, AmLocale.Alert(),
AmDialog.B_OK, AmLocale.Base_language_delete());
■ baaskeele kustutamine *
dialoogi tulemuse töötlemine
true

Lisa 6.
Klassi *Item14Listener* struktuur

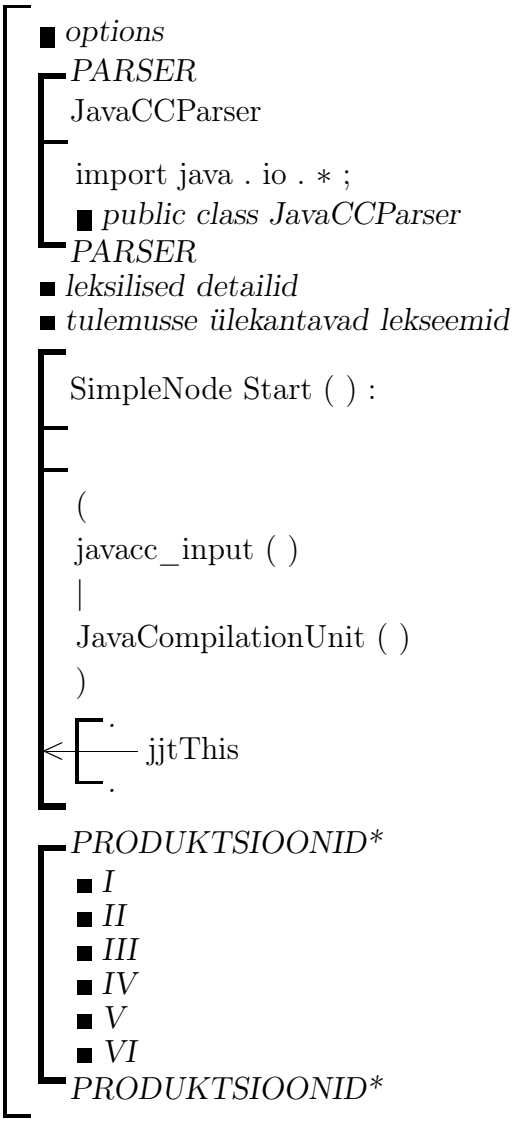
import java.awt.∗;
import java.awt.event.∗;
import javax.swing.∗;
import javax.swing.event.∗;
import java.io.∗;
import java.lang.∗;

class Item14Listener implements ActionListener - - - *Restore BaseLanguage*

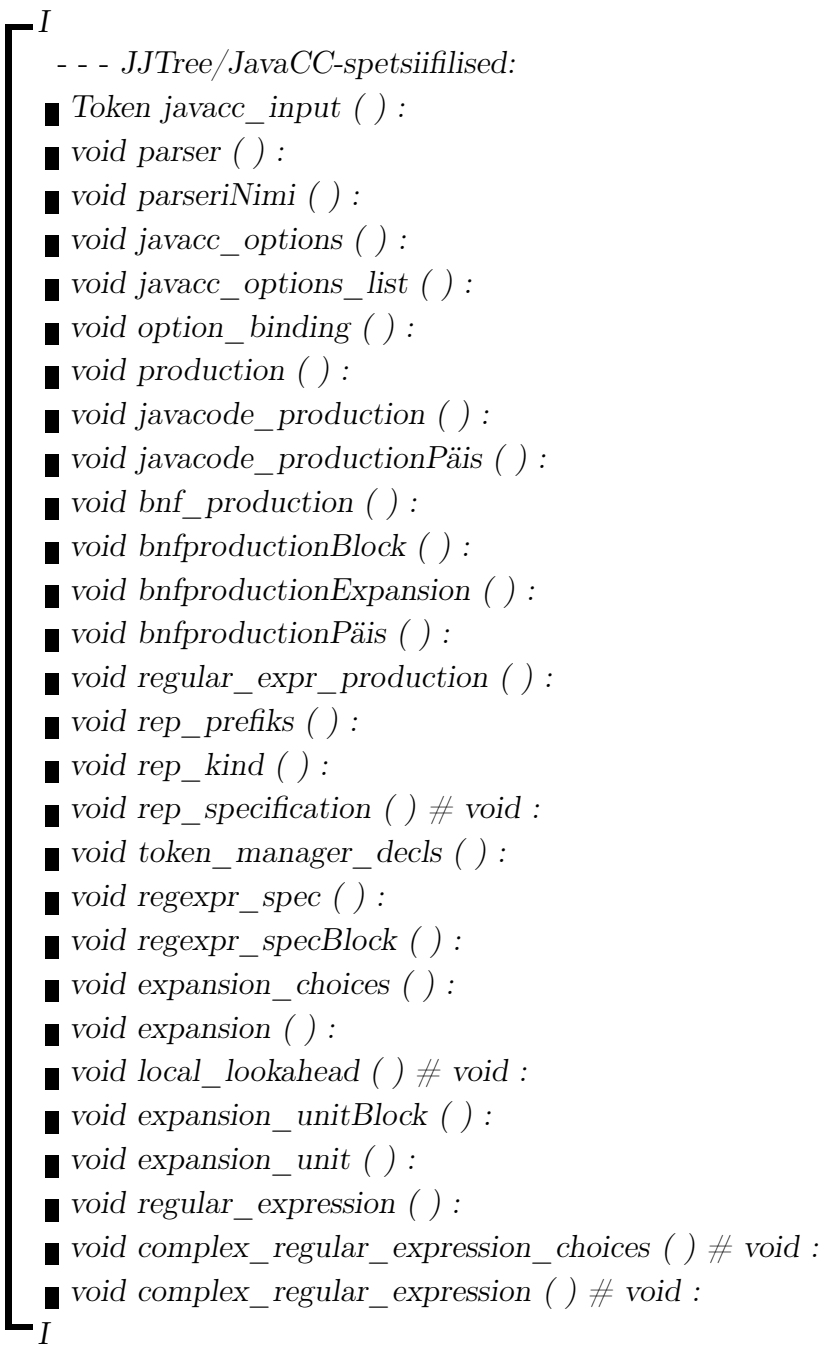
AmFrame frame;;
String title;;

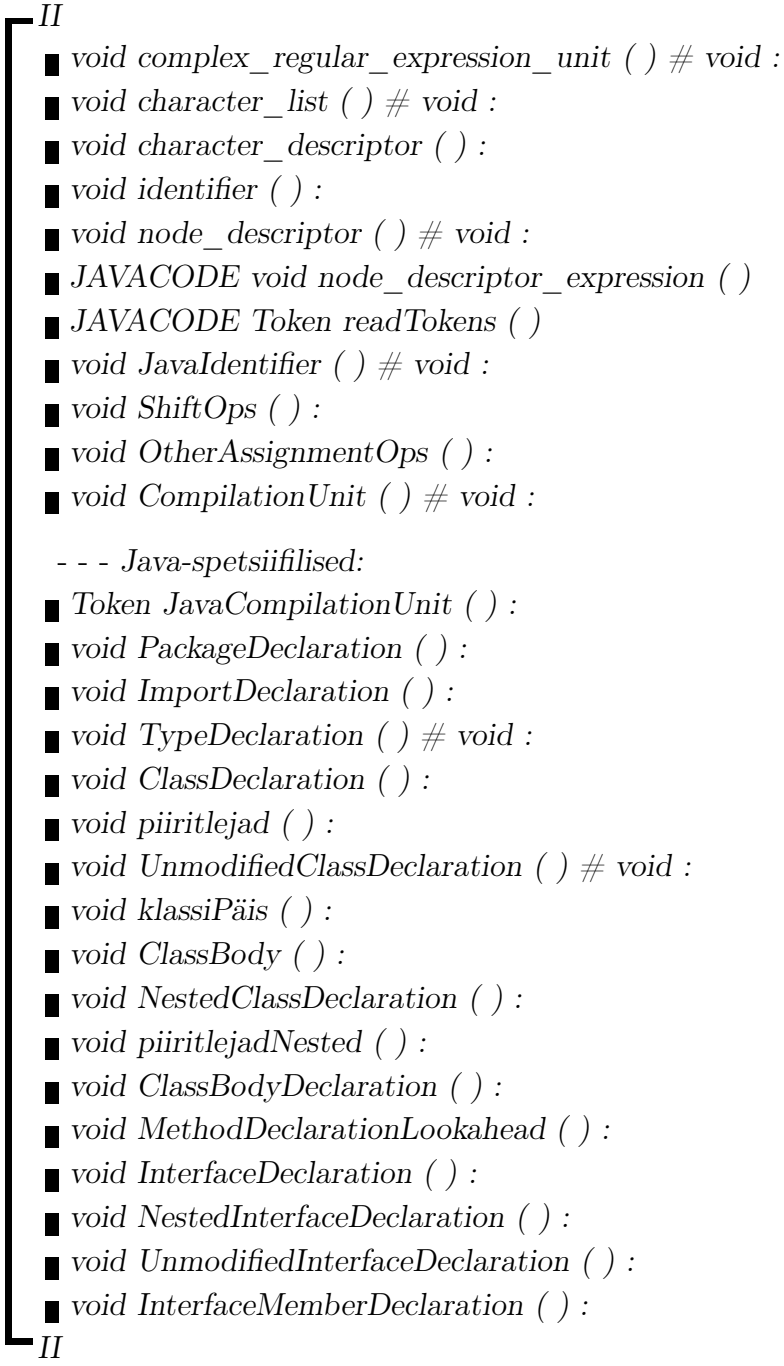
- *Item14Listener(AmFrame fr)*
- *public static void restoreLangFiles(String title)*
- *public static int lang_nr(String title)*
- *public static boolean lang_inwork(String title)*
- *public static void rcopyFile(String kust, String kuhu)*
- *public static int lang_restore(String fdir)*
- *public static void removeLangFiles(String title)*
- *public void actionPerformed(ActionEvent event)*

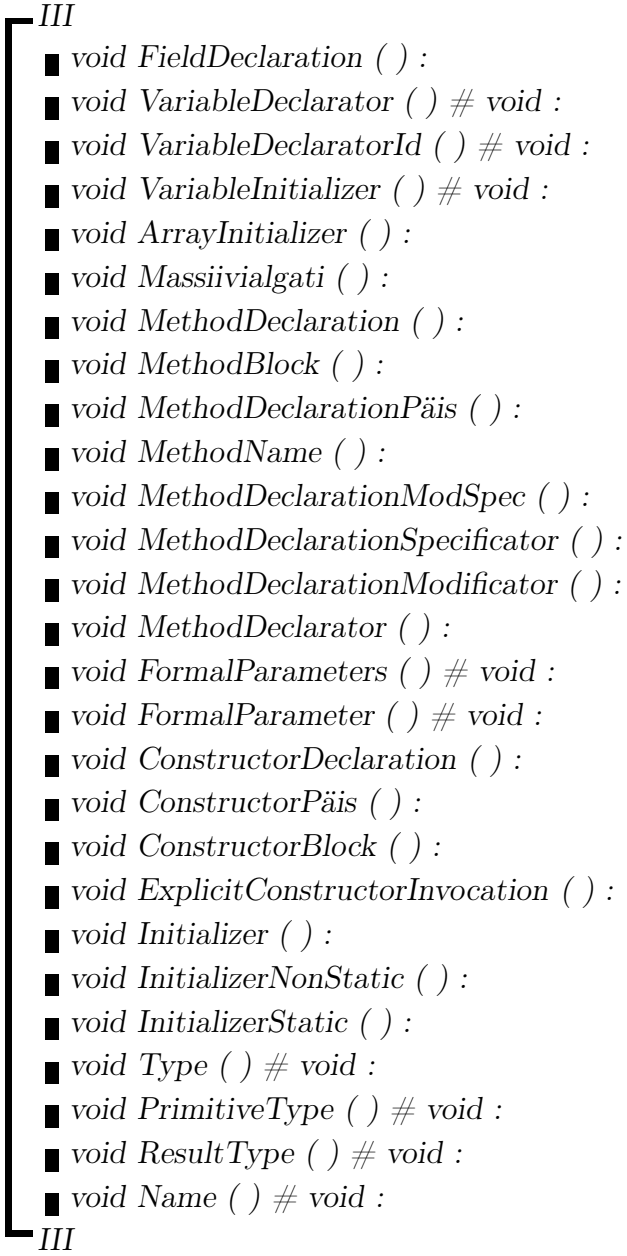
Lisa 7.
JavaCC grammatikafail

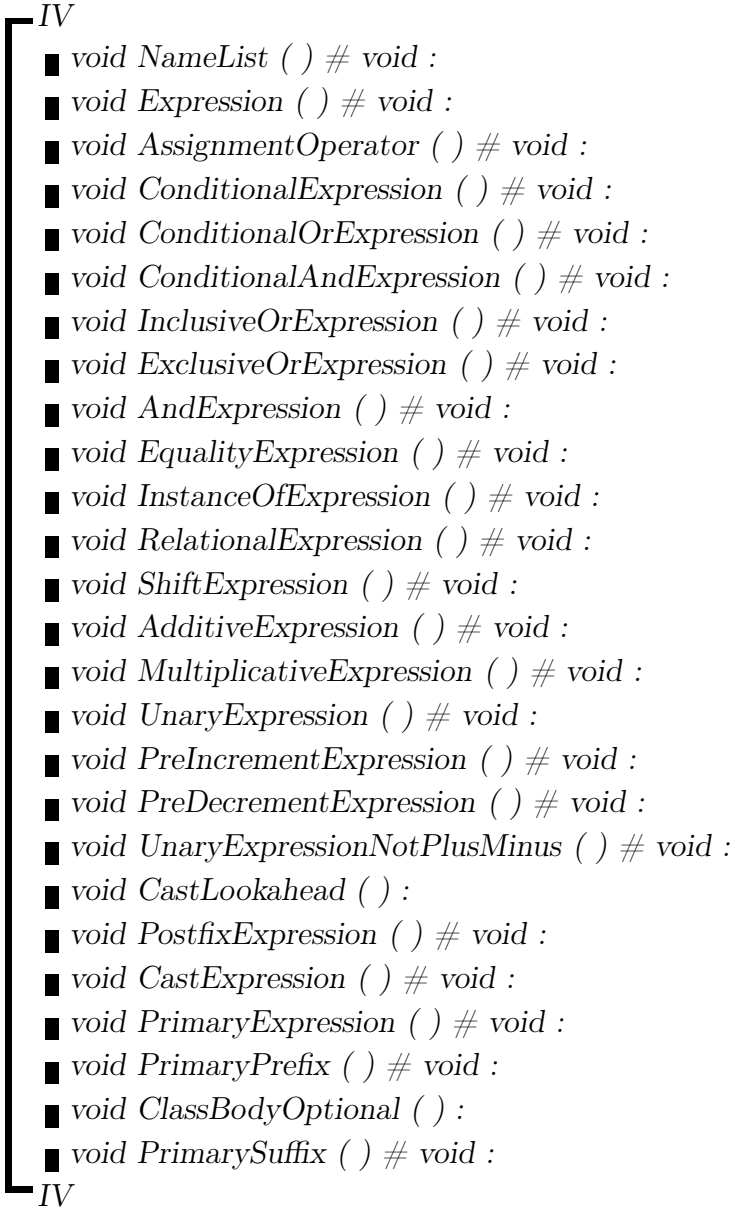


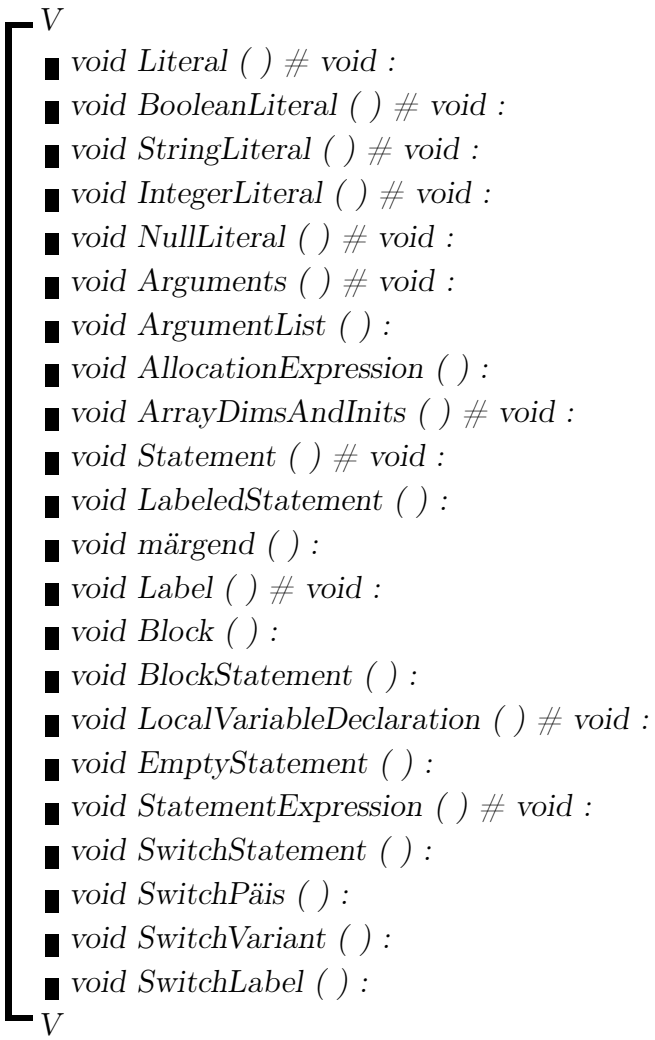
PRODUKTSIOONID


I
- - - JJTree/JavaCC-spetsiifilised:
Token javacc_input ( ) :
void parser ( ) :
void parseriNimi ( ) :
void javacc_options ( ) :
void javacc_options_list ( ) :
void option_binding ( ) :
void production ( ) :
void javacode_production ( ) :
void javacode_productionPäis ( ) :
void bnf_production ( ) :
void bnfproductionBlock ( ) :
void bnfproductionExpansion ( ) :
void bnfproductionPäis ( ) :
void regular_expr_production ( ) :
void rep_prefiks ( ) :
void rep_kind ( ) :
void rep_specification ( ) # void :
void token_manager_decls ( ) :
void regexpr_spec ( ) :
void regexpr_specBlock ( ) :
void expansion_choices ( ) :
void expansion ( ) :
void local_lookahead ( ) # void :
void expansion_unitBlock ( ) :
void expansion_unit ( ) :
void regular_expression ( ) :
void complex_regular_expression_choices ( ) # void :
void complex_regular_expression ( ) # void :
I


II
void complex_regular_expression_unit ( ) # void :
void character_list ( ) # void :
void character_descriptor ( ) :
void identifier ( ) :
void node_descriptor ( ) # void :
JAVACODE void node_descriptor_expression ( )
JAVACODE Token readTokens ( )
void JavaIdentifier ( ) # void :
void ShiftOps ( ) :
void OtherAssignmentOps ( ) :
void CompilationUnit ( ) # void :
- - - Java-spetsiifilised:
Token JavaCompilationUnit ( ) :
void PackageDeclaration ( ) :
void ImportDeclaration ( ) :
void TypeDeclaration ( ) # void :
void ClassDeclaration ( ) :
void piiritlejad ( ) :
void UnmodifiedClassDeclaration ( ) # void :
void klassiPäis ( ) :
void ClassBody ( ) :
void NestedClassDeclaration ( ) :
void piiritlejadNested ( ) :
void ClassBodyDeclaration ( ) :
void MethodDeclarationLookahead ( ) :
void InterfaceDeclaration ( ) :
void NestedInterfaceDeclaration ( ) :
void UnmodifiedInterfaceDeclaration ( ) :
void InterfaceMemberDeclaration ( ) :
II

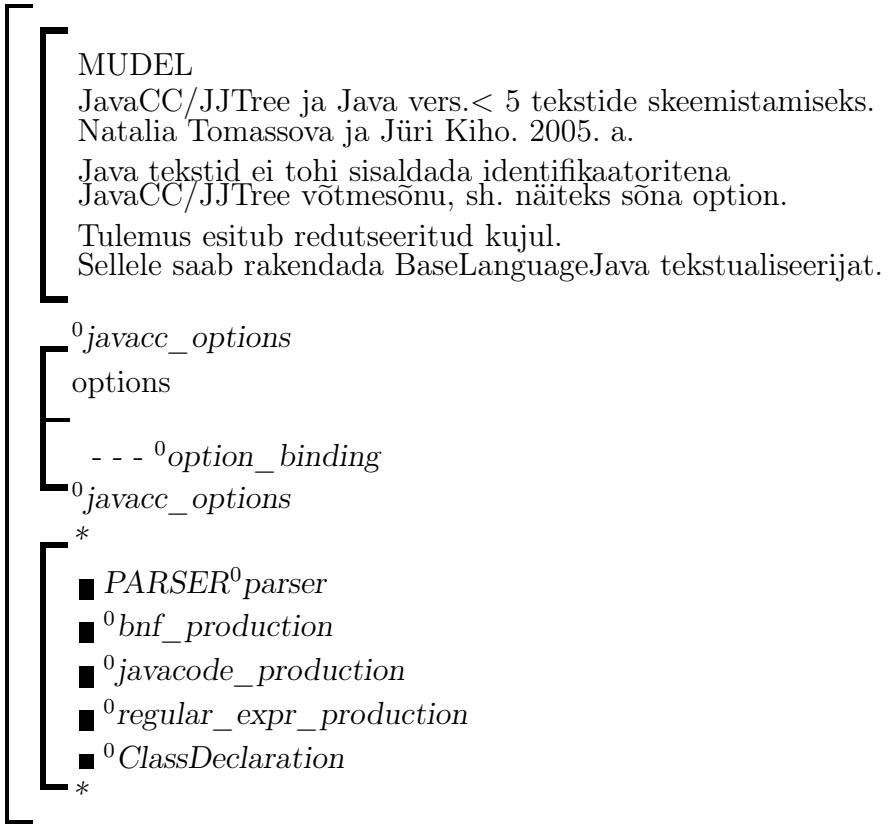

III
void FieldDeclaration ( ) :
void VariableDeclarator ( ) # void :
void VariableDeclaratorId ( ) # void :
void VariableInitializer ( ) # void :
void ArrayInitializer ( ) :
void Massiivialgati ( ) :
void MethodDeclaration ( ) :
void MethodBlock ( ) :
void MethodDeclarationPäis ( ) :
void MethodName ( ) :
void MethodDeclarationModSpec ( ) :
void MethodDeclarationSpecificator ( ) :
void MethodDeclarationModificator ( ) :
void MethodDeclarator ( ) :
void FormalParameters ( ) # void :
void FormalParameter ( ) # void :
void ConstructorDeclaration ( ) :
void ConstructorPäis ( ) :
void ConstructorBlock ( ) :
void ExplicitConstructorInvocation ( ) :
void Initializer ( ) :
void InitializerNonStatic ( ) :
void InitializerStatic ( ) :
void Type ( ) # void :
void PrimitiveType ( ) # void :
void ResultType ( ) # void :
void Name ( ) # void :
III

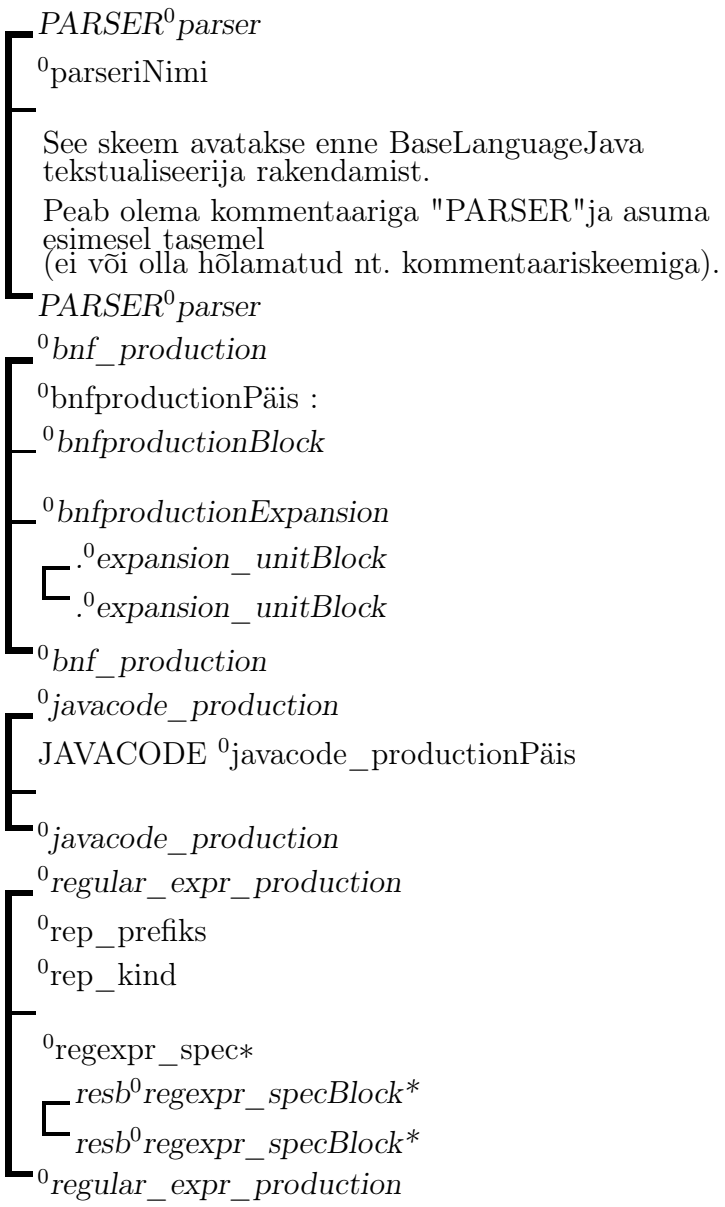

IV
void NameList ( ) # void :
void Expression ( ) # void :
void AssignmentOperator ( ) # void :
void ConditionalExpression ( ) # void :
void ConditionalOrExpression ( ) # void :
void ConditionalAndExpression ( ) # void :
void InclusiveOrExpression ( ) # void :
void ExclusiveOrExpression ( ) # void :
void AndExpression ( ) # void :
void EqualityExpression ( ) # void :
void InstanceOfExpression ( ) # void :
void RelationalExpression ( ) # void :
void ShiftExpression ( ) # void :
void AdditiveExpression ( ) # void :
void MultiplicativeExpression ( ) # void :
void UnaryExpression ( ) # void :
void PreIncrementExpression ( ) # void :
void PreDecrementExpression ( ) # void :
void UnaryExpressionNotPlusMinus ( ) # void :
void CastLookahead ( ) :
void PostfixExpression ( ) # void :
void CastExpression ( ) # void :
void PrimaryExpression ( ) # void :
void PrimaryPrefix ( ) # void :
void ClassBodyOptional ( ) :
void PrimarySuffix ( ) # void :
IV

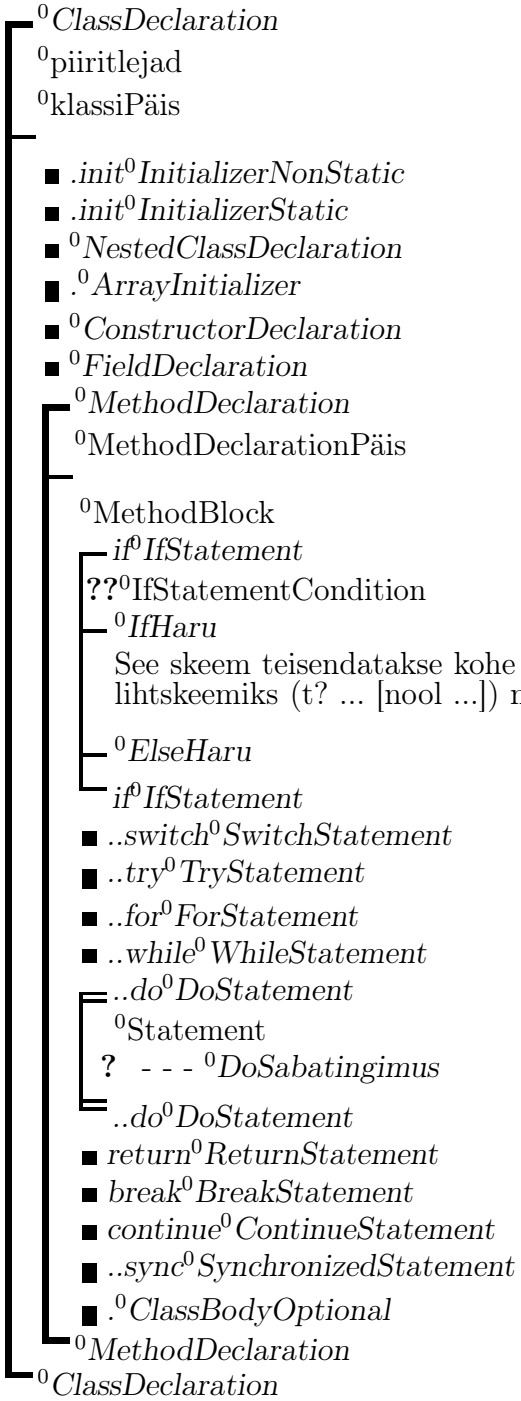

V
void Literal ( ) # void :
void BooleanLiteral ( ) # void :
void StringLiteral ( ) # void :
void IntegerLiteral ( ) # void :
void NullLiteral ( ) # void :
void Arguments ( ) # void :
void ArgumentList ( ) :
void AllocationExpression ( ) :
void ArrayDimsAndInits ( ) # void :
void Statement ( ) # void :
void LabeledStatement ( ) :
void märgend ( ) :
void Label ( ) # void :
void Block ( ) :
void BlockStatement ( ) :
void LocalVariableDeclaration ( ) # void :
void EmptyStatement ( ) :
void StatementExpression ( ) # void :
void SwitchStatement ( ) :
void SwitchPäis ( ) :
void SwitchVariant ( ) :
void SwitchLabel ( ) :
V

- *VI*
  - *void IfStatement ( ) :*
  - *void IfHaru ( ) :*
  - *void ElseHaru ( ) :*
  - *void IfStatementCondition ( ) :*
  - *void IfStatementBlock ( ) :*
  - *void IfStatementElse ( ) :*
  - *void WhileStatement ( ) :*
  - *void WhileStatementCondition ( ) :*
  - *void WhileStatementBlock ( ) :*
  - *void DoStatement ( ) :*
  - *void DoSabatingimus ( ) :*
  - *void ForStatement ( ) :*
  - *void ForPäis ( ) :*
  - *void ForStat ( ) :*
  - *void ForInit ( ) # void :*
  - *void StatementExpressionList ( ) :*
  - *void ForUpdate ( ) # void :*
  - *void BreakStatement ( ) :*
  - *void BreakStatementBlock ( ) :*
  - *void ContinueStatement ( ) :*
  - *void ContinueStatementBlock ( ) :*
  - *void ReturnStatement ( ) :*
  - *void ReturnStatementBlock ( ) :*
  - *void ThrowStatement ( ) :*
  - *void SynchronizedStatement ( ) :*
  - *void SynchronizedPäis ( ) :*
  - *void TryStatement ( ) :*
  - *void TryBlock ( ) :*
  - *void TryCatch ( ) :*
  - *void TryCatchPäis ( ) :*
  - *void TryFinally ( ) :*
  - *void AssertStatement ( ) :*
- *VI*

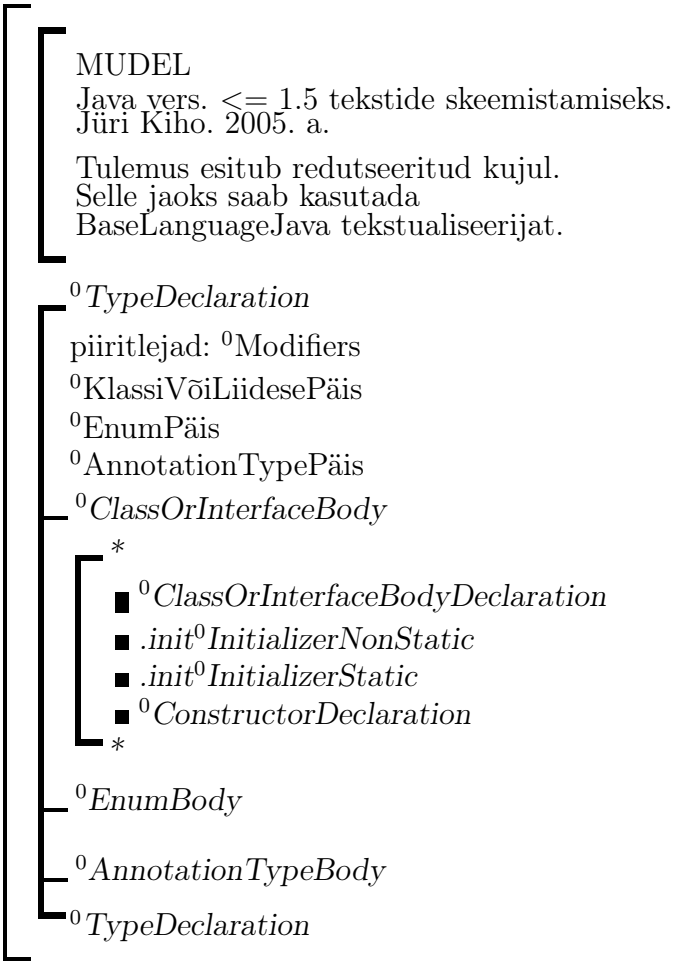
Lisa 8. *JavaCC* skeemmudel

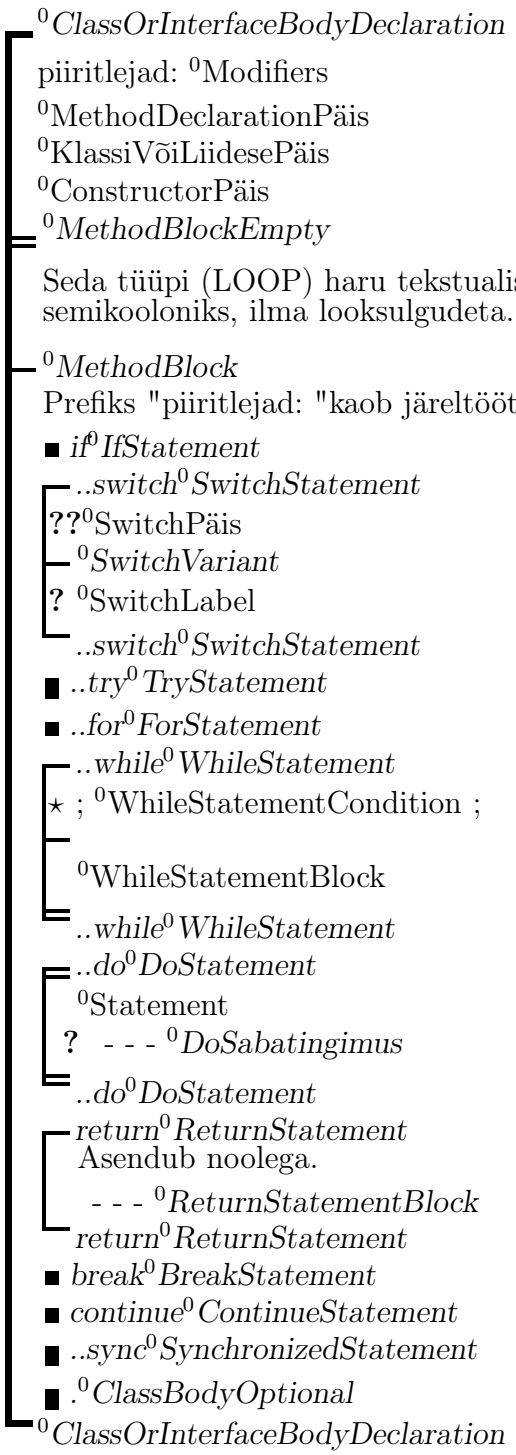


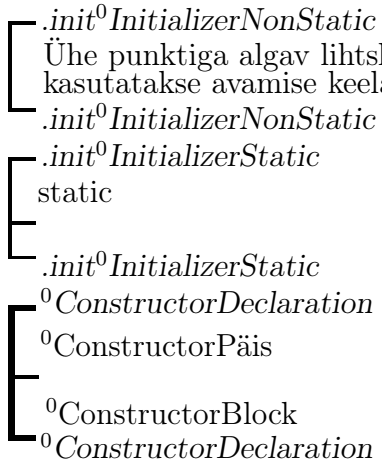

PARSER⁰parser
⁰parseriNimi
See skeem avatakse enne BaseLanguageJava
tekstualiseerija rakendamist.
Peab olema kommentaariga "PARSER"ja asuma
esimesel tasemel
(ei või olla hõlamatud nt. kommentaariskeemiga).
PARSER⁰parser
⁰bnf_production
⁰bnfproductionPäis :
⁰bnfproductionBlock
⁰bnfproductionExpansion
.⁰expansion_unitBlock
.⁰expansion_unitBlock
⁰bnf_production
⁰javacode_production
JAVACODE ⁰javacode_productionPäis
⁰javacode_production
⁰regular_expr_production
⁰rep_prefiks
⁰rep_kind
⁰regexpr_spec∗
resb⁰regexpr_specBlock*
resb⁰regexpr_specBlock*
⁰regular_expr_production


0ClassDeclaration
0piiritlejad
0klassiPäis
.init0InitializerNonStatic
.init0InitializerStatic
0NestedClassDeclaration
.0ArrayInitializer
0ConstructorDeclaration
0FieldDeclaration
0MethodDeclaration
0MethodDeclarationPäis
0MethodBlock
if0IfStatement
??0IfStatementCondition
0IfHaru
See skeem teisendatakse kohe pärast skeemistamist lihtskeemiks (t? ... [nool ...]) märgendiga "..if".
0ElseHaru
if0IfStatement
..switch0SwitchStatement
..try0TryStatement
..for0ForStatement
..while0WhileStatement
..do0DoStatement
0Statement
? - - - 0DoSabatingimus
..do0DoStatement
return0ReturnStatement
break0BreakStatement
continue0ContinueStatement
..sync0SynchronizedStatement
.0ClassBodyOptional
0MethodDeclaration
0ClassDeclaration

Lisa 9. *Java5* skeemmudel




0ClassOrInterfaceBodyDeclaration
piiritlejad: 0Modifiers
0MethodDeclarationPäis
0KlassiVõiLiidesePäis
0ConstructorPäis
0MethodBlockEmpty
Seda tüüpi (LOOP) haru tekstualiseeritakse
semikooloniks, ilma looksulgudeta.
0MethodBlock
Prefiks "piiritlejad: "kaob järeltöötlusel
if0IfStatement
..switch0SwitchStatement
??0SwitchPäis
0SwitchVariant
? 0SwitchLabel
..switch0SwitchStatement
..try0TryStatement
..for0ForStatement
..while0WhileStatement
⋆ ; 0WhileStatementCondition ;
0WhileStatementBlock
..while0WhileStatement
..do0DoStatement
0Statement
? - - - 0DoSabatingimus
..do0DoStatement
return0ReturnStatement
Asendub noolega.
- - - 0ReturnStatementBlock
return0ReturnStatement
break0BreakStatement
continue0ContinueStatement
..sync0SynchronizedStatement
.0ClassBodyOptional
0ClassOrInterfaceBodyDeclaration


.init0InitializerNonStatic
Ühe punktiga algav lihtskeemi kommentaari
kasutatakse avamise keelamiseks redutseerimisel.
.init0InitializerNonStatic
.init0InitializerStatic
static
.init0InitializerStatic
0ConstructorDeclaration
0ConstructorPäis
0ConstructorBlock
0ConstructorDeclaration

Lisa 10.
Keele *PSE JJTree*-grammatika lihtteksti kujul

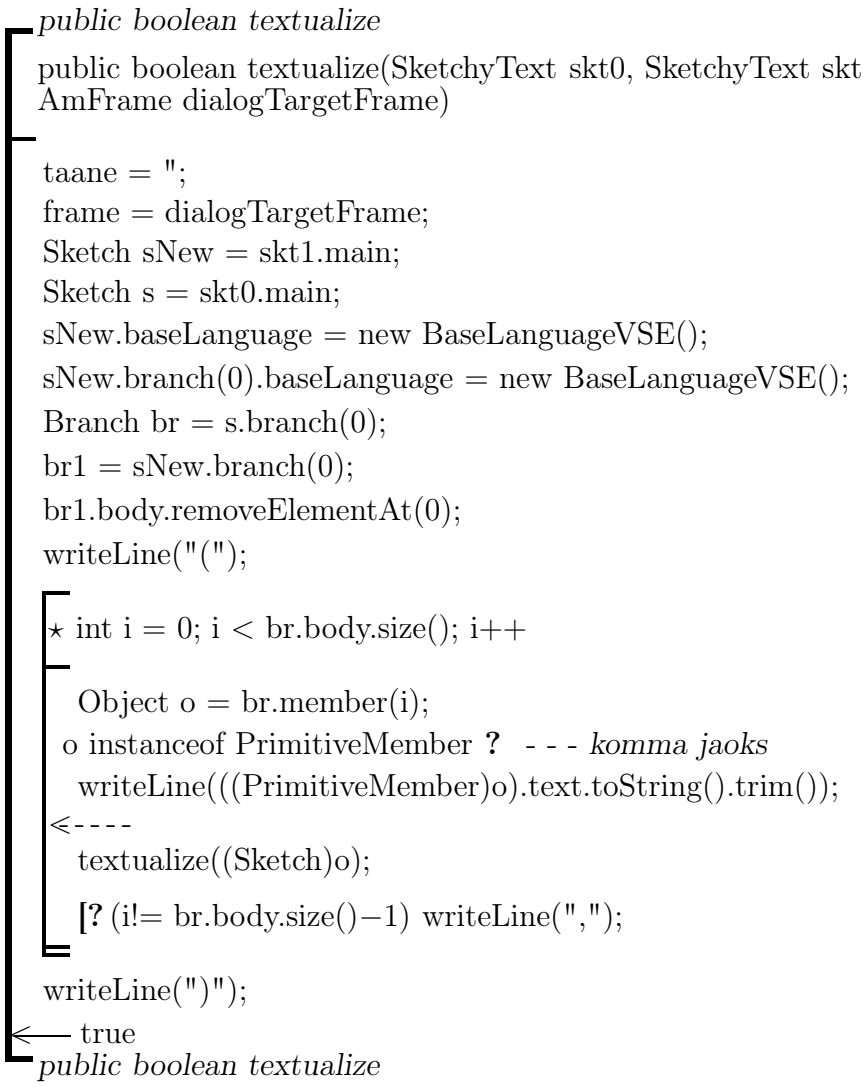
```
options { JAVA_UNICODE_ESCAPE = true; STATIC = false; }

PARSER_BEGIN(PSEParser)

import java.io.*;

public class PSEParser {
static Token t;
static Sketch parse(String inputFileName) throws ParseException,
FileNotFoundException {
PSEParser parser = new PSEParser(new FileReader(inputFileName));
SimpleNode t = parser.Start();
Sketch parsipuu = BaseLanguageParseTree.toSketch
(t, PSEParserTreeConstants.jjtNodename);
return(parsipuu);
}

PARSER_END(PSEParser)

SPECIAL_TOKEN : {
" "
| "\t"
```

```
| "\r"
| "\f"
| "\n"
}

/* IDENTIFIERS */
TOKEN :
{
< IDENTIFIER: <LETTER> (<LETTER> | <DIGIT>)* >
| < # LETTER: [ "A-"Z","_", "a"- "z"] >
| < # DIGIT: [ "0"- "9"] >
}

/* SEPARATORS */
TOKEN : {
< LPAREN: "(" >
| < RPAREN: ")" >
| < COMMA: "," >
}

TOKEN :
{
< ANYTHING_ELSE: ( ~ [ ] ) >
}

/*productions-produktsioonid*/
void tIdentifier(): {}{
t = <IDENTIFIER>
{jjtThis.addText(t.image);}}

/** THE GRAMMAR SPECIFICATION STARTS HERE **/
SimpleNode Start() : {} // algussümbol
{
metsaPSE()
```

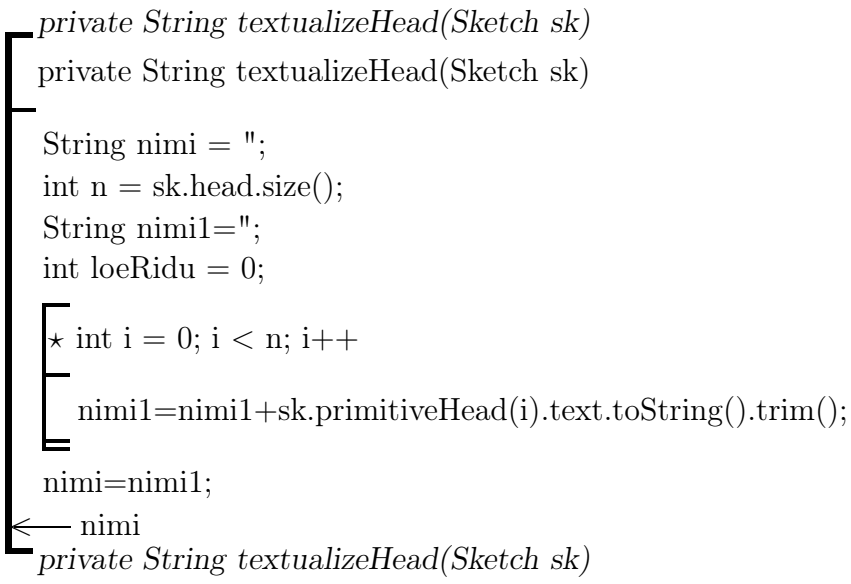
```
<EOF>
{ return jjtThis; }
}
Token metsaPSE() #void:
{
Token t = getToken(1);
}
{
"("puuPSE() (","puuPSE())* ")"
{ return t;}
}
void puuPSE():
{}
{
[ metsaPSE()] juur()
}
void juur() :
{}
{
(tIdentifier())
}
```

Lisa 11.
Klass *BaseLanguagePSE*

```
import java.util.*;
import java.io.*;

class BaseLanguagePSE extends BaseLanguage

static AmFrame frame;
■ BaseLanguagePSE()
■ PARSE
■ SKETCHIFY
■ NORMALIZE
TEXTUALIZE
    ⋆ static

    String nimi=";

    ■ public boolean textualize *
    ■ private void textualize(Sketch sk)*
    ■ private String textualizeHead(Sketch sk)*
    private void writeLine(String str)*
    private void writeLine(String str)

    Text newT = new Text();
    newT.rows.removeElementAt(0);
    newT.rows.addElement(new Row(taane + str, Default.foregroundColor));
    br1.body.addElement(new PrimitiveMember(Primitive.SIMPLE, newT));
    private void writeLine(String str)*
TEXTUALIZE
■ prepareTex
■ nextPrimitiveHeadType
```


public boolean textualize
public boolean textualize(SketchyText skt0, SketchyText skt1,
AmFrame dialogTargetFrame)
taane = ";
frame = dialogTargetFrame;
Sketch sNew = skt1.main;
Sketch s = skt0.main;
sNew.baseLanguage = new BaseLanguageVSE();
sNew.branch(0).baseLanguage = new BaseLanguageVSE();
Branch br = s.branch(0);
br1 = sNew.branch(0);
br1.body.removeElementAt(0);
writeLine("(");
⋆ int i = 0; i < br.body.size(); i++
Object o = br.member(i);
o instanceof PrimitiveMember ? - - - komma jaoks
writeLine(((PrimitiveMember)o).text.toString().trim());
⩽----
textualize((Sketch)o);
[? (i!= br.body.size()−1) writeLine(",");
writeLine(")");
⟸ true
public boolean textualize

```
private void textualize(Sketch sk)
  int loeRidu = 0;
  ⋆ int i = 0; i < sk.body.size(); i++
    Branch br = ((Branch)(sk.member(i)));
    [? (br.body.size()!= 0) writeLine("(");
      ⋆ int j = 0, n = br.body.size(); j < n; j++
        Object o = br.member(j);
       o instanceof Sketch ?
        textualize((Sketch)o);
        [? (j!=n−1) writeLine(",");
    [? (br.body.size()!=0) writeLine(")");
    loeRidu++;
  String nimi = textualizeHead(sk);
  writeLine(nimi);
```


private String textualizeHead(Sketch sk)
private String textualizeHead(Sketch sk)
String nimi = "";
int n = sk.head.size();
String nimi1="";
int loeRidu = 0;
⋆ int i = 0; i < n; i++
nimi1=nimi1+sk.primitiveHead(i).text.toString().trim();
nimi=nimi1;
nimi
private String textualizeHead(Sketch sk)

Lisa 12. *AmadeusJSN* ja käesoleva magistritöö e-koopia

Köitele lisatud CD sisaldab järgmisi faile:

1. *AmadeusJSN.zip* – süsteemi *Amadeus* viimane versioon (mai 2005).
2. *MagToo.zip* – käesoleva magistritöö e-koopia.
3. *readme.txt* – täiendav abi-info CD kasutajale.