T A R T U Ü L I K O O L
MATEMAATIKA-INFORMAATIKATEADUSKOND
Arvutiteaduse instituut
Tarkvarasüsteemide õppetool

Svetlana Golovko

Projekti mõiste ja teostus süsteemis *Amadeus*

Magistritöö (40 ap)

Juhendaja: prof. Jüri Kiho

TARTU 2004

Sisukord

Sissejuhatus

Käesolev töö jätkab Tartu Ülikoolis kavandatud süsteemi *Amadeus* edasiarendamist ja täiustamist.

Üldiselt süsteemi *Amadeus* kujutab endast tarkvaraarenduse keskkonda, kus töödeldavad dokumendid on esitatud ühtsel skeemkujul, nn. skeemtekstidena.

Süsteemis *Amadeus*, nii nagu teisteski arendussüsteemides, lisanduvad faili (teksti/koodi) toimetamisele mitmesugused eel- ja järeltegevused. Polüfunktsionaalses süsteemis *Amadeus* on mitmeid erinevaid failide avamise ja salvestamise funktsioone. Tavalised avamise (*Open*) ja salvestamise (*Save*) funktsioonid aga töö aluseks olevas lähteversioonis puuduvad. Kasutaja peab ise otsustama, missugust funktsiooni (või funktsioonide jada) ta kasutab antud faili avamisel või salvestamisel. Ühe failiga töötamisel ei ole see probleemiks, kuid tööfailide arvu kasvamisel võib see osutuda üsna häirivaks.

Iga integreeritud arenduskeskkond peab andma võimaluse mugavalt kasutada kõiki süsteemi poolt pakutud funktsioone. Samuti on ka süsteemi *Amadeus* jaoks vaja vahendit (nn. *projekti*), mille abil saaks lihtsustada olemasolevate töötlusvahendite sagedast kasutamist. Projektitugi muudab rutiinse töö süsteemis *Amadeus* kahtlemata mugavamaks ja kiiremaks.

Projektile esitavad nõuded on järgmised.

- Projekt peab võimaldama organiseerida tööd failide rühmadega (nt. suurte, paljudest eri osadest koosnevate tööde puhul).
- Projekt peab siduma faili tegevuste toimetamise eel- ja järeltegevuste jadaga, eeskätt seonduvalt faili avamise ja salvestamise tegevustega.
- Projekt peab tagama tegevuste jadade automaatse rakendumise.

Töö teoreetilisemat laadi ülesandeks on selgitada, kuidas käsitletakse projekte teistes arendussüsteemides ja, sellest lähtuvalt, defineerida süsteemi *Amadeus* jaoks projekti mõiste. Praktilise teostuse osas tuleb valida projekti *Amadeus* arenduskeskkonnas esitamise viis ja realiseerida projekti mõiste süsteemis *Amadeus*. Töö koosneb järgmistest sisulistest põhiosadest

1. Projekti mõiste
   Selles peatükis antakse ülevaade projektide realiseerimisest ja nende võimalustest erinevates arenduskeskkondades. Tuuakse välja nende erinevusi ja sarnasusi.

2. Projekt süsteemis *Amadeus*
   Selles peatükis antakse lühike ülevaade süsteemist *Amadeus*, kirjeldatakse rutiinset tööd süsteemis *Amadeus*, selgitatakse, milliseid tegevusi tüüpiliselt korratakse iga faili jaoks. Samuti on esitatud ühe lihtsama arendustöö projekti näide (tudengi semestritöö). Järgnevalt on antud projekti mõiste definitsioon süsteemi *Amadeus* jaoks.

3. Projekti teostus süsteemis *Amadeus*
   Selle peatüki alguses esitatakse teostuse nõudeid täpsustav kasutajavaate kirjeldus. Käsitlemist leiavad realisatsioon põhiklassid *ActionController* ja *ProjectBoard*.

Lisadeks on CD-l esitatud süsteemi *Amadeus* uus versioon *AmadeusJS*, käesoleva töö tekst ja näidisprojektikirjeldused, mida autor kasutas süsteemi *Amadeus* arendamisel ning antud magistritöö teksti koostamisel.

Peatükk 1

Projekti mõiste

1.1 Mis on projekt

> Projekt - planeeritud töö või tegevuse osa, mille eesmärgiks on konkreetne tulemus [*Cambridge Advanced Learner's Dictionary*].

Mõiste *projekt* on kasutusel enamikus integreeritud arenduskeskkondades (ingl. k. *integrated development environment*, IDE). Epigraaf annab küll sõna *projekt* üldtuntud seletuse, kuid ühegi IDE projekti puhul see tegelikult ei sobi. Samas tuleb nentida, et reeglina polegi konkreetse IDE korral projekti mõiste täpselt määratletud, vähemalt mitte IDE abi-info (*Help*) dokumentatsioonis. Tüüpiliselt hakatakse kohe andma selgitusi, kuidas kasutada projekti antud keskkonnas. Seega määratletakse IDE projekti mõiste kaudselt, projekti funktsionaalsuse kaudu.

Arendussüsteemi töökeskkonna efektiivne organiseerimine nõuab võimalust ühendada erinevaid objekte ühe mõiste (projekt) alla. Ilmselt ongi IDE projekti üheks olulisemaks funktsiooniks kasutatavate ressursside koondamine hõlpsamini käideldavatesse rühmadesse. Ühendatavad objektid võivad olla täiesti erinevat laadi. Lihtsamaks ühendamise eesmärgiks on tagada tihtikasutatavate objektide kiire kättesaamine töökeskkonna seansi ajal. Samuti on grupeerimine oluline siis, kui objektid on ehitusblokkideks suuremas

süsteemis, mille ülesehitamiseks rakendatakse mitmeid erinevaid protsesse. Rühma moodustavad nt. objektid, millele rakendatakse samu protsesse.

Projektiga hõlmatavateks objektideks võivad olla tekstifailid, pildid jt. ressursid. Samuti võib projekt sisaldada rakenduse ehitamiseks vajalikku infot. Projekti töö organisatsioon ja realisatsioon on töökeskkonniti üsnagi erinev.

Objektide sidumiseks mingi projektiga on kaks võimalust:

- objektide asukoht määrab nende projekti kuulumise või mittekuulumise,
- projekt ise kirjeldab, missugused objektid projekti kuuluvad ja kus nad asuvad.

Teine variant on rohkem levinud, kuna see on paindlikum ja mugavam. Kuid leidub süsteeme, kus esimene variant on teise variandiga kombineeritud. Mõne IDE korral saab mitu projekti ühendada ühte töötsooni. Projektid ühes töötsoonis ei pea olema üksteisega seotud. Andmete detailsem grupeerimine (näiteks tüübi järgi) projekti sees ei ole tavaline, aga on siiski üks võimalikke projekti organisatsiooni liike.

Vaatamata arenduskeskkondade ja nende projektikäsitluse erinevustele leidub ka sarnasusi mitmete projektirealisatsioonide vahel. Tihti kuulub projektile töökeskkonna mingi ekraaniosa, kus kuvatakse kõik projekti kuuluvate failide nimed. Topeltklõps faili nimel avab faili. Projekti seaded on tavaliselt muudetavad selleks eraldi ettenähtud dialoogiaknas.

Lihtsa IDE projekti realisatsiooni näiteks on redaktor EditPlus [4]. Redaktor sisaldab kausta akent (*directory window*), kus on võimalik kuvada nii mingi draivi kaustade struktuur kui ka valitud kausta sisu. Redaktori projekti all mõeldakse ainult mingit failide kogumit. Projekt on aga esitatav kui eri kaust (füüsiliselt failisüsteemis mitte eksisteeriv) kausta aknas. Faili avamiseks piisab kaksikklõpsust kausta aknas faili nimel.

1.2 Ülevaade projektidest mõnedes teistes arenduskeskkondades

Käesolevas jaotises vaadeldakse projekti mõistet mõnedes tuntumates arenduskeskkondades. Võimalusel püütakse välja tuua järgmised momendid projekti töö organiseerimise kohta:

1. Kuidas defineeritakse abi-info dokumentatsioonis projekti mõiste.
2. Projektide kirjeldamise vorm.
3. Failide projekti kuuluvuse kirjeldamine. Projektifailide asukoht (kas näiteks sama kohas, kus on projektikirjelduse fail, või saab ressursse kaasata teistest kaustadest).
4. Failide grupeerimise võimalused.
5. Erivõimalused.

Vaatlusele võetud arenduskeskkonnad on valitud võimalikult mitmekesised, nimelt Eclipse platvorm, Java, C++ ja Pascal keelte keskkonnad ning üks TEX/LATEX failide toimeti. Põhiinformatsioon iga süsteemi kohta on saadud nende abi-info dokumentatsioonist.

1.2.1 Eclipse

Eclipse [6] on universaalne platvorm "kõige ja eriti mitte millegi" jaoks. Eclipse'i eriväärtus peitub *plug-in* vahendites, mis "õpetavad" Eclipse'i, kuidas töötada Java failidega, veebi lehtedega, graafikaga, videoga. Eclipse'i abil saab samuti luua vahendid, mis integreeruvad teiste vahenditega nii sujuvalt, et ei ole märgatagi, kus üks vahend lõpeb ja teine algab.

1. Iga projekt sisaldab failide ja kaustade hulka. Neid objekte nimetatakse *ressurssideks*. Ka projekti ennast nimetatakse ressursiks. Eclipse'i dokumentatsioonis leidub info, kuidas luua uus projekt, kuidas importida ja eksportida faile, luua uusi katalooge projektis jne.

2. Igale projektile vastab failisüsteemis oma projekti kaust (sama nimega, mis on projektil). Vaikimisi paigutatakse kõik projektide kaustad Eclipse'i installeerimise kausta alamkausta *workspace*. Projekti kirjeldav *xml* formaadis fail ".project" asub projekti kaustas. Tabelis 1.1 leidub Eclipse'i lihtsa projekti faili näide.

```
<?xml version="1.0" encoding="UTF-8"?>
<projectDescription>
   <name>Simple Project</name>
   <comment>This is a season project.</comment>
   <projects>
     <project>org.seasons.sdt</project>
     <project>ColStuff</project>
   </projects>
   <buildSpec>
    </buildSpec>
   <natures>
   </natures>
</projectDescription>
```

Tabel 1.1: Eclipse projekti kirjeldav fail.

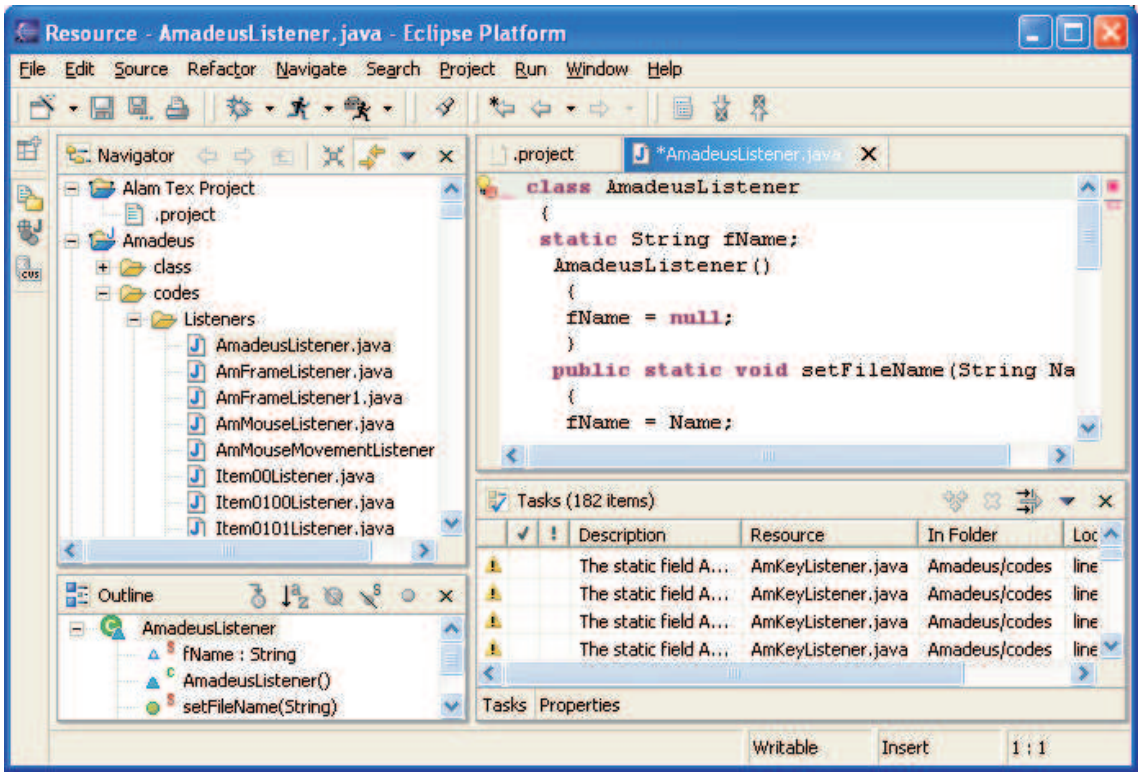
Antud juhul on projekti nimeks *Simple Project* ja projekt sisaldab kahte viidet teistele projektidele. Lisaks ".project" failile Eclipse saab luua ka *xml* formaadis teise faili ".classpath", mis hoiab infot projektis kasutatavate teede kohta.

3. Kõik kaustad ja failid, mis asuvad failisüsteemi projekti kaustas, on vaikimisi selle projekti osadeks. Nagu näha tabelist 1.1, ei ole Eclipse projekti failis täpselt kirjas millest projekt koosneb. Samuti saab projektis Eclipse IDE kaudu luua nö. link-kaustad ja failid, mille asukoht erineb projekti kaustast. Link-kaustad määratakse projekti failis elemendina *linkedResources* (märgiste <linkedResources> ja </linkedResources> vahel).

4. Projektifaile saab grupeerida alamkaustadesse. Need alamkaustad luuakse ka füüsiliselt failisüsteemis.

5. Projekti kustutamisel keskkonnast saab failisüsteemist kustutada ka kogu projekti sisu. Projekti saab "sulgeda", mis tähendab, et projekti osad (ressursid, projektifailid) ei ole nähtavad ja kättesaadavad IDE-s, kuid projektikirjeldus jääb ikka nähtavaks keskkonna navigeerimise aknas. Nagu öeldud Eclipse'i dokumentatsioonis, nõuavad suletud projektid vähem mälu.

Eclipse IDE akna pilt on kujutatud joonisel 1.1.



Joonis 1.1: Eclipse akna pilt.

1.2.2 Microsoft Visual C++ 6.0

Microsoft Visual C++ [5] on graafiline kasutajaliides C++ keeles programmeerimiseks.

1. Projekt on seotud failide rühm. Tavaliselt sisaldab see kõik vajalikud failid mingi tarkvarakomponendi arendamiseks.

2. Keskkonna projektid hoitakse töötsoonis. Töötsoon võib sisaldada mitut projekti. Ainult üks töötsoon saab olla avatud antud ajahetkel. Projekti loomisel luuakse failisüsteemis kaust projekti nimega, kuhu salvestatakse kõik projektifailid. Projektis on võimalik luua alamprojekte. Alamprojekt on projekt, millel on seos mõne teise projektiga. Iga projekt saab olla suvalise teise projekti alamprojektiks. Kuid töötsoonis peab olema vähemalt üks projekt, mis ei ole ühegi teise projekti alamprojekt.

3. Failid võivad olla projekti lisatud erinevatest failisüsteemi kohtadest. Projekti failideks on

   - <töötsooni_nimi>.dsw - töötsooni (*workspace*) fail.

     Töötsooni failis on iga projekt määratud oma nimega ja oma *dsp* faili asukohaga. Igas projektis on kirjas alamprojektide nimed.

     Näiteks tabelis 1.2 on toodud ühe *dsw* faili osa, mis defineerib töötsooni projekti *Core*. Viimases on projekt *CPPExtension* defineeritud kui projekti *Core* alamprojekt.

```
Project: "Core\"=.\Core\Core.dsp - Package Owner=<4>
Package=<5>
{{{
}}}
Package=<4>
{{{
Begin Project Dependency
Project_Dep_Name CPPExtension
End Project Dependency
}}}
```

Tabel 1.2: Microsoft Visual C++ *dsw* faili osa.

   - <projekti_nimi>.dsp - ühe projekti fail.

Ridade

 # Begin Project ja

 # End Project

vahel on kirjas info kogu projekti kohta – projekti osad, osade sõltuvused ja terve projekti ning selle osade omadused. Omadused esitatakse kujul:

 # PROP [BASE] 'omaduse nimetus' 'omaduse väärtus'

Iga omadus kuulub kas kogu projektile või siis failide grupile. Esimesel juhul ta on kirjas kohe rea *# Begin Project* järel, teisel aga kohe peale projekti grupi defineerimist.
Omaduste näiteid:

 # PROP Output_ Dir "Release"

 # PROP Intermediate_ Dir "Release"

 # PROP Ignore_ Export_ Lib 0

 # PROP Default_ Filter "h;hpp;hxx;hm;inl"

Projekti konfiguratsioonist sõltuvad omadused kirjeldatakse eraldi blokkides. Ridade

 # Begin Target

 # End Target

vahel paiknevad failide ja gruppide kirjeldused. Failid on jaotatud gruppideks (kaustad IDE töötsoonis). Kaustad töötsoonis ei ole tegelikud kaustad failisüsteemis. Ridade

 # Begin Group "Grupi nimi" ja

 # End Group

vahel on kirjas grupi omadused ja gruppi kuuluvate failide kirjeldused kujul

 # Begin Source File

SOURCE=<kaustatee>\<failinimi>

 # End Source File

kus kaustatee on määratud *dsp* faili asukohta suhtes. Näiteks:

SOURCE=..\..\STEP1\CHILDFRM.CPP

või

SOURCE=..\MyClas.cpp.

Peale ülalkirjeldatu sisaldab *dsp* veel rohkesti informatsiooni projekti kompileerimiseks ja ehitamiseks, seda infot ei tohi kasutaja "käsitsi", st. IDE väliselt muuta.

4. Projektifailid grupeeritakse (keskkonnas, kuid mitte failisüsteemis, luuakse uued kaustad projekti kausta all).

5. Projektiga on seotud mõiste *konfiguratsioon*. Konfiguratsioon määrab C++ projekti ehitamise omadused. Kasutaja saab muuta olemasolevate konfiguratsioonide omadusi või luua oma konfiguratsiooni mingi olemasoleva konfiguratsiooni põhjal. Samuti saab kasutaja valida projekti ehitamiseks ühe enda poolt loodud konfiguratsioonidest, vaikimisi konfiguratsiooni või rakendada korraga mitut konfiguratsiooni ühes ehitamise operatsioonis. Konfiguratsioonide omadused võivad olla määratud projekti või faili tasemel. Esimesed on rakendatavad igale projektifailile. Projekti konfiguratsiooni omadus on rakendatav failile seni, kuni see ei ole ümber defineeritud mingis failitasemel konfiguratsioonis. Konfiguratsioonide omadused on kirjas projekti failis.

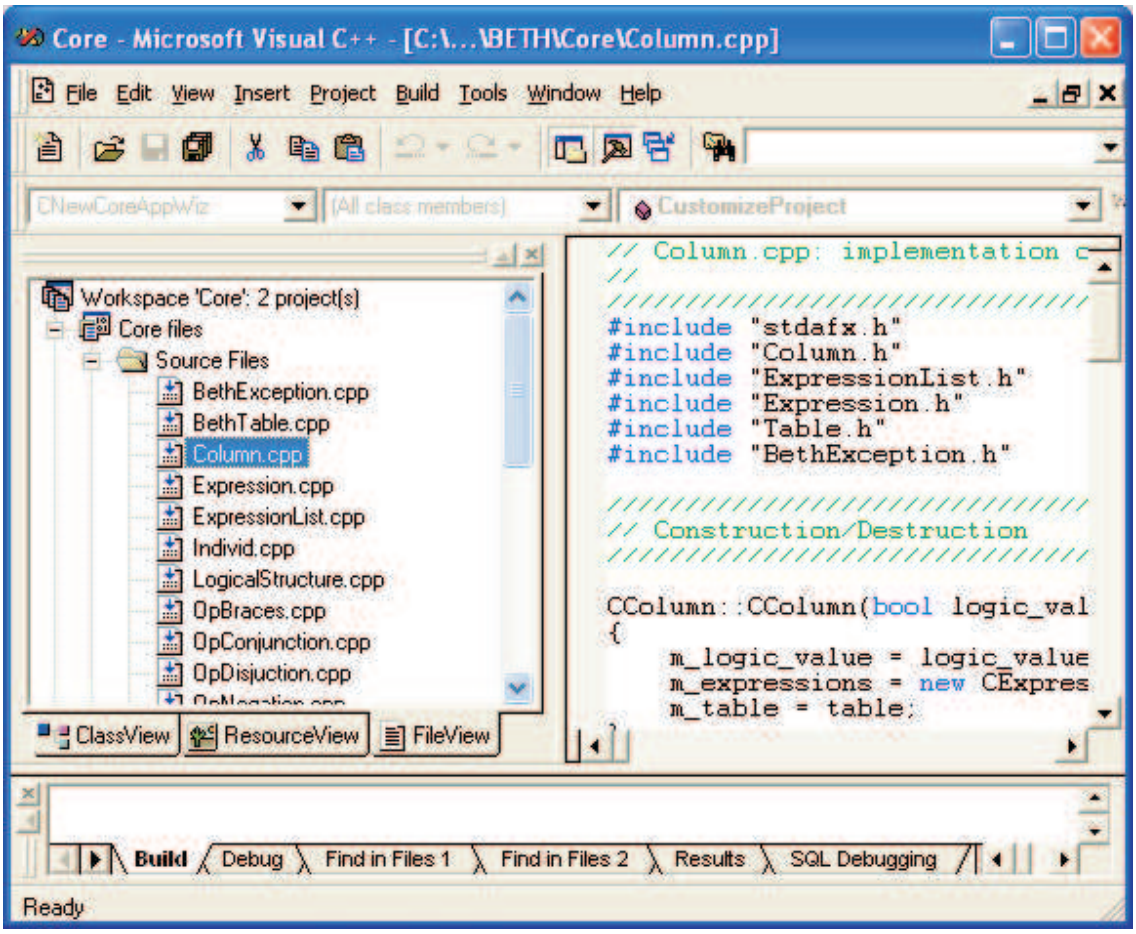
Microsoft Visual C++ keskkonna akna pilt leidub joonisel 1.2.

1.2.3 JCreator

JCreator [7] on graafiline kasutajaliides programmeerimiseks keeles *Java*.

1. Projekt on *Java* rakendus või paketiteek (*package library*). Projekti töötsoon JCreator keskkonnas on kaust, kus hoitakse informatsiooni kogu projekti kohta. Kui projekt on loodud, siis JCreator paneb programmi lähteteksti ja teised failid projekti töötsooni kausta. Samasse töötsooni saab panna ka alamprojektid. Alamprojekte saab kasutada pakettide jaoks, mis on seotud kasutaja *Java* rakendusega, või lihtsalt JDK lähtetekstide failide hoidmiseks.

2. Seansi ajal hoitakse IDE projektid töötsoonis. Töötsoon võib sisaldada mitut projekti. Ainult üks töötsoon saab olla avatud antud ajahetkel. Projekti loomisel luuakse failisüsteemis kaust kasutaja poolt antud nimega (vaikimisi projekti nimi), kuhu salvestatakse kõik projektifailid.



Joonis 1.2: Microsoft Visual C++ akna pilt.

3. Kõik IDE-s loodud kaustad luuakse samuti failisüsteemis. Töötsooni erikaustas *External Files* saab hoida projekti kausta väliseid faile.

   Projekti failideks on

   - *<töötsooni_nimi>.jcu* ja *<töötsooni_nimi>.jcw* - töötsooni failid.

     Failis *<töötsooni_nimi>.jcu* on kirjas töötsooni eelmise sulgemise aegne seis. Näiteks: missugune projekt oli aktiivne, missugused failid projektidest olid avatud ja missugune nendest oli aktiivne,

missuguste projektide all olid kaustad.

Failis *<töötsooni_ nimi>.jcw* on kirjas töötsooni üldine info - töötsooni nimi, töötsooni projektid ja samuti viit vastavale *jcu* failile.

Mõlemad failid on *xml* formaadis. Näide *jcu* failist on esitatud tabelis 1.3, näide *jcw* failist aga tabelis 1.4.

```
<?xml?>
<workspace_user>
    <active_project>
        <label>Help</label>
    </active_project>
    <projects>
        <project>
            <label>Am</label>
            <showfiles>false</showfiles>
            <folderitem>
                <path>Am\src</path>
                <ext/>
            </folderitem>
            <folderitem>
                <path>Am\src\Listeners</path>
                 ...
    </projects>
    <open_files>
        <docitem>
            <path>Am\src\Am.java</path>
            <source>true</source>
            <index>0</index>
            <active>true</active>
        </docitem>
        <docitem>
            <path>Am\src\AmFrame.java</path>
            <source>true</source>
            <index>1</index>
            <active>false</active>
        </docitem>
    </open_files>
</workspace_user>
```

Tabel 1.3: JCreator *jcu* fail.

```
<?xml?>
<workspace>
        <settings>
                <label>Am</label>
        </settings>
        <users>
                <user>
                        <path>Am.jcu</path>
                </user>
        </users>
        <projects>
                <project>
                        <path>Help\Help.jcp</path>
                </project>
                <project>
                        <path>Am\Am.jcp</path>
                </project>
        </projects>
</workspace>
```

Tabel 1.4: JCreator *jcw* fail.

- *<projekti_ nimi>.jcp* - ühe projekti fail.

  Fail *jcp*, samuti nagu *jcu* ja *jcw*, on *xml* formaadis. Fail sisaldab informatsiooni projekti mõnede omaduste kohta (näiteks kompileerimise ja käivitamise kohta) ning projekti kuuluvate failide ja kaustade kohta. Projekti omadused on kirjeldatud järgmiste märgiste vahel:

  <settings></settings>,

  <runtimes></runtimes>,

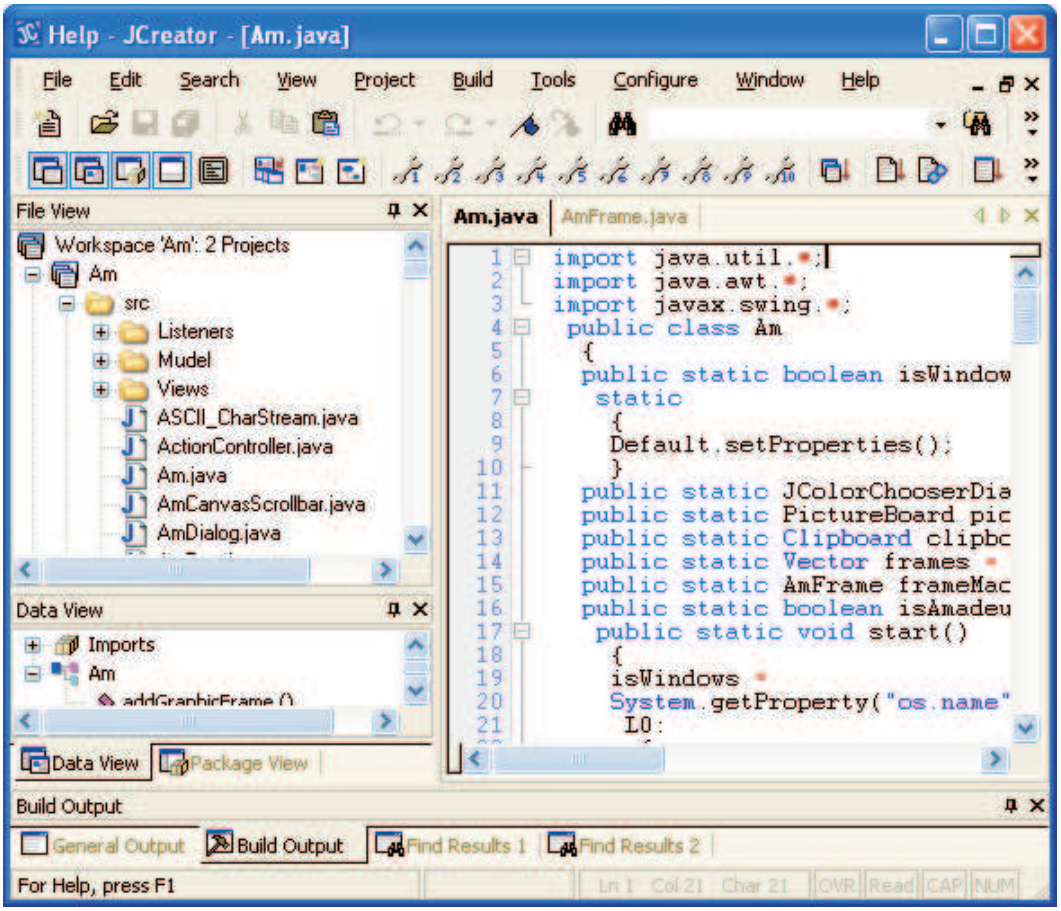
  <libraries></libraries>.

  Märgiste <files> ja </files> vahel on kirjeldatud projektis olevate kaustade ja failide omadused. Iga fail projektis on esitatud märgistega <fileitem> ja </fileitem>.

4. Faile saab grupeerida alamprojektidesse või lihtsalt projekti alamkaustadesse. Need alamkaustad luuakse ka füüsiliselt failisüsteemis.

5. Projekti kustutamisel keskkonnast (projekti töötsoonist kustutamisel) jääb projekti kaust failisüsteemis paigale ja kustutatud projekti sisu näidatakse *External Files* all.

JCreator IDE akna pilt on joonisel 1.3.



Joonis 1.3: JCreator IDE akna pilt.

1.2.4 Delphi

Delphi on graafiline kasutajaliides Pascal keeles programmeerimiseks. Siinkohal tugineme versioonile Borland Delphi Enterprise 5.0 [8].

1. Projekt on failide kogu, mis moodustab mingi rakenduse või dünaamilise teegi. Mõned neist failidest on loodud rakenduse kirjutamise ajal, teised aga rakenduse kompileerimise ajal. Projekte võib ühendada projektigruppidesse.

2. Projekti lähtekoodi keskne koht on projekti fail. Delphi muudab seda faili vastava rakenduse arendamisel. Projekti fail sisaldab viitu kõikidele vormidele ja üksustele (*unit*), mida kasutatakse rakenduses. Projekti faili nime laiendiks on tavaliselt *dpr*. Näiteks tabelis 1.5 on toodud ühe projekti *dpr* fail.

```
program Project1;                          { projekti identifikaator }
uses                                { projektis kasutatud üksused... }
  Forms,                                    { ...vormita üksused... }
  Main_form in ´Main_form.pas´{fCalculator};
                                           { ...ja vormiga üksused. }
{$R .RES}                                  { kasutatavad ressursid }
begin                                            { põhiploki algus }
  Application.Initialize;
  Application.Title := 'Kalkulaator';
  Application.CreateForm(TfCalculator,Calculator);
                                 { automaatselt luua esimene vorm }
  Application.Run;                          { käivitada rakendus }
end.                                              { põhiploki lõpp }
```

Tabel 1.5: Delphi projekti kirjeldatav fail.

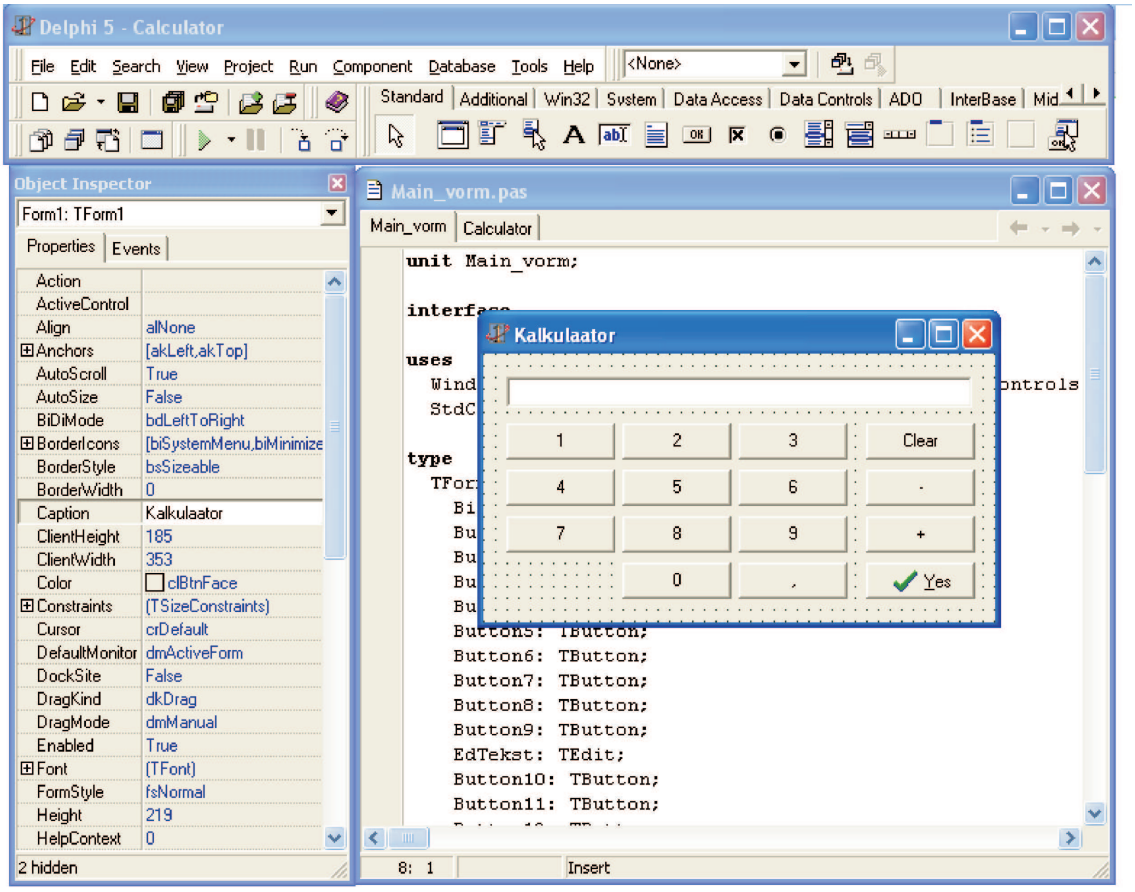
   Samuti sisaldab projekti fail rakenduse põhiplokki (reserveeritud sõnade *begin* ja *end* vahel). Põhiplokis initsialiseeritakse ja käivitatakse rakendus, laetakse mällu vajalikud vormid ja defineeritakse rakenduse põhiomadused. Projekti faili sisu saab vaadata IDE-s, kuid muuta seda ei soovitata. Projekti muutmine toimub alati Delphi liidese kaudu.

3. Projekti failid võivad asuda erinevates kaustades.

4. Ei ole võimalust grupeerida faile ühe projekti alla.

5. Kompileerimist puudutavad omadused ei ole määratud projektis, vaid keskkonnas. See tähendab kord seatuna kehtivad need omadused iga avatud projekti jaoks.

Delphi akna pilt on joonisel 1.4.



Joonis 1.4: Delphi akna pilt.

1.2.5 WinShell

WinShell [9] on graafiline kasutajaliides LaTeX või TeX dokumentidega töötamiseks.

1. Autoril ei õnnestunud leida abi-info dokumentatsioonis midagi, mis meenutaks projekti definitsiooni.

2. Failid projektis ei pea omavahel olema kuidagi loogiliselt seotud. Ainuke, mis on kohustuslik - projektis peab olema määratud peadokument. Keskkonnas avatud projektid ja projektide failinimed näidatakse projekti ruumis (*Projectspace*). Projekt ruumis saab lisada faile projekti, kustutada faile projektist, muuta projekti peadokumenti. On olemas erimenüü *Project*, kus asuvad projektiga seotud põhitegevuste valikud (luua uus projekt, avada projekt, sulgeda projekt, salvestada projekt, salvestada projekt teise nimi all). Projekti fail (laiendiga *wsp*) sisaldab relatiivseid teenimesid (teenimed on antud projekti faili asukohta suhtes). See ongi antud keskkonnas projekti mõiste kaudne definitsioon.

3. Näide projekti failist on toodud tabelis 1.6.

[main]
.\MinuPeafail.tex
[texfiles]
.\Projekt.tex
.\Sissejuhatus.tex
..\..\tiitel.tex

Tabel 1.6: WinShell projekti kirjeldatav fail.

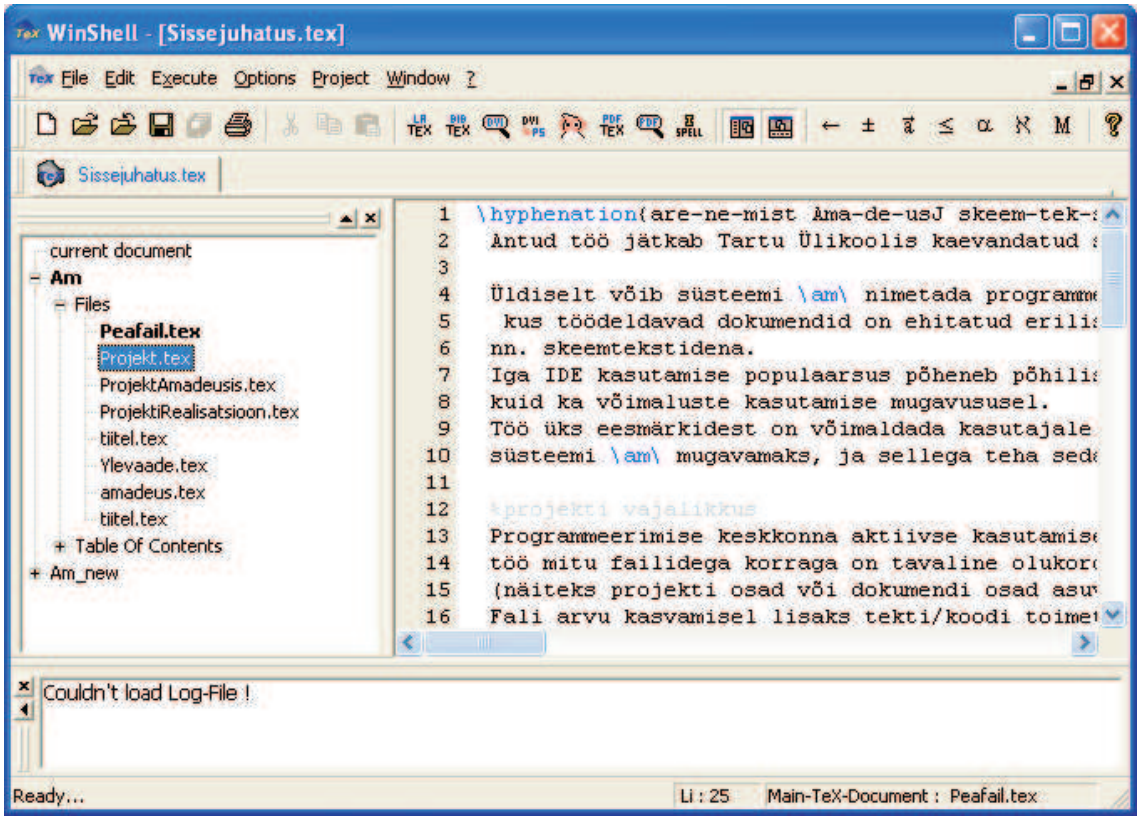
   WinShell keskkonnas võib olla avatud mitu projekti korraga. Ainult üks nendest saab olla aktiivne antud ajahetkel.

4. Faile ei saa grupeerida.

WinShelli akna pilt leidub joonisel 1.5.

* * *

Esitatud ülevaade kinnitab veelgi tõdemust, et erinevates arenduskeskkondades mõistetakse projekti all üsnagi erinevaid objektide ja atribuutide



Joonis 1.5: WinShelli akna pilt.

komplekte. Seetõttu on arusaadav, et IDE projekti üldmõiste määratlust ei ole lihtne formuleerida. Olemasolevate IDE projektide ühise joonena saab esile tuua asjaolu, et projektid võimaldavad ressursse (faile) kas füüsiliselt või siis virtuaalselt grupeerida. Grupeerimise kriteeriumiks aga on failide asukohad, ühisomadused või nende sarnased funktsionaalsed atribuudid.

Üldises plaanis võib öelda, et iga konkreetne IDE projekt hõlmab kahte sorti faile – projektikirjeldust sisaldavad failid ja projektiga kaasatud objektfailid ehk ressursid. Viimaseid nimetame edaspidi *projektifailideks*. Projektikirjelduse faili nimetame lühemalt ka *projekti failiks*.

Võrdlemisi kirju on pilt ka selles osas, millist keelt (süntaksit) kasutatakse projektikirjelduste esitamiseks. Projekti faili sisu esitakse kas spetsiaalses keeles (Microsoft Visual C++, WinShell), *xml* vormis (Eclipse, JCreator)

või töödeldavate projektifailide keeles (Delphi). Tõsi küll, projektikirjelduse konkreetne tekst on enamasti varjestatud graafilise kasutajaliidesega.

Kui projektifaili avamine (hiireklõpsu abil) toimub enamuses keskkondades üsna sarnaselt, siis projektifailide töötlemise protseduurid kirjeldatakse erinevates arenduskeskkondades vägagi erinevalt.

Peatükk 2

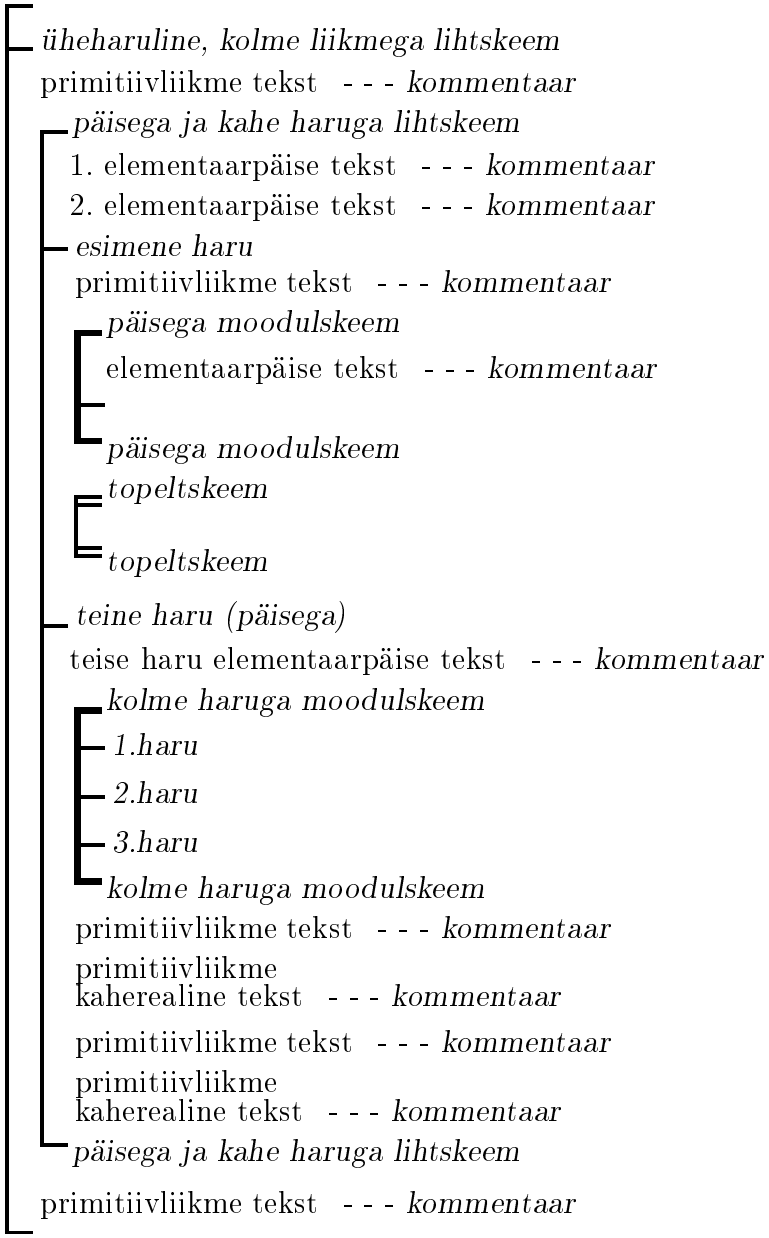
Projekt süsteemis *Amadeus*

2.1 Lühiülevaade süsteemist *Amadeus*

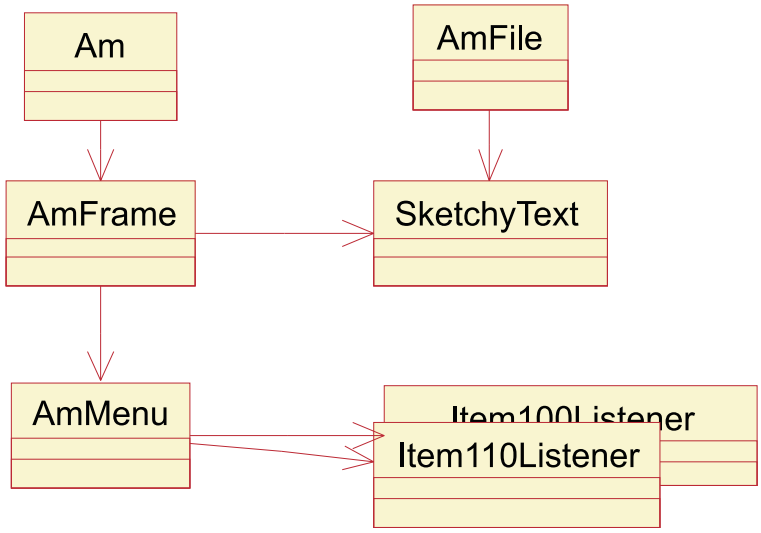
Amadeus on programmeerimiskeeles *Java* teostatud mitmevaateline mitmekeelne üldfunktsionaalne süsteem tarkvaraliste tekstidega (programmide lähtekoodid, LATEX dokumendid, HTML tekstid) töötamiseks. *Amadeus* toetab tekstide graafilist esitamist ja toimetamist märksa sügavamas plaanis, kui tavalised tekstitoimetid. Iga süsteemis *Amadeus* töödeldav tekst kujundatakse skeemtekstina. Skeemteksti mõiste täpne kirjeldus leidub raamatus [3]. Siinkohal piirdume vaid ühe skeemteksti kuju selgitava näitega joonisel 2.1.

Töö süsteemis *Amadeus* toimub üksteisest sõltumatutes ekraaniakendes, ehk raamides. Ühe skeemteksti jaoks on üks aken; mõned skeemtekstile rakendatud süsteemi funktsioonid võivad luua uusi aknaid ja kajastada seal funktsiooni töö tulemusena saadud/muudetud skeemteksti. Situatsioon, kus on avatud mitu akent korraga, on tüüpiline süsteemi *Amadeus* jaoks.

Skeem joonisel 2.2 näitab, kuidas põhiklassid *AmFile*, *SketchyText*, *AmFrame* on omavahel seotud. Iga faili sisu esitatakse skeemtekstina (*SketchyText*). Skeemtekst kuvatakse klassi *AmFrame* poolt tehtud raamis. Kõik süsteemi funktsioonid saab välja kutsuda raami menüü (*AmMenu*) abil.



Joonis 2.1: Skeemteksti näitestruktuur.



Joonis 2.2: Skeem süsteemi *Amadeus* klassidest (*Am, AmFrame, AmMenu, SketchyText, AmFile).*

2.1.1 Süsteemi *Amadeus* funktsionaalsus

Süsteemi lähteversioonis on realiseeritud järgmised toimetamise ja töötlemise funktsioonid:

Toimetamise funktsioonid

- Skeemteksti elementide sisestamine/lisamine skeemtekstisse
- Skeemteksti elementide kustutamine skeemtekstist
- Kopeerimise, lõikamise, kleepimise jmt. funktsioonid
- Skeemteksti elementide tüübi muutmine
- Skeemteksti osa haaramine lihtskeemiga

Töötlemise funktsioonid (sõltuvad jooksva skeemteksti baaskeelest)

- Skeemistamine
- Redutseerimine

- Normaliseerimine
- Teksti baaskeele muutmine
- TEX-ettevalmistus
- Skeemteksti süsteemi sisestamine:
  - Uus
  - Importida (fail mõnes *Amadeus* väljundformaadis)
  - Importida makro
  - Lugeda tekst (tavaline tekst)
- Skeemteksti teisendamine väljund formaati:
  - Eksportida liht-HTML
  - Eksportida HTML
  - Eksportida liht-XML
  - Eksportida XML
  - Printida liht-TEX
  - Printida PostScript
  - Kirjutada tekst

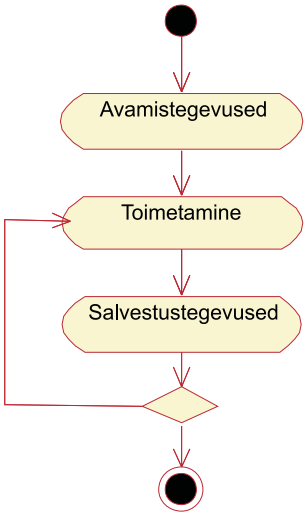
Süsteemi *Amadeus* lähteversioonis ei ole tavapäraseid üldtöötlemise funktsioone *Open* ja *Save*. Nende asemel peab kasutaja ise otsustama, millist süsteemi vahendit ta tahab faili avamiseks rakendada ja milliseid vahendeid ta kasutab skeemteksti salvestamiseks.

Edaspidi, rääkides menüüfunktsioonist või -käsust mõeldakse selle all mingit töötlusfunktsiooni.

2.1.2 Rutiinne töö skeemtekstiga süsteemis *Amadeus*

Töö failiga toimub tavaliselt joonisel 2.3 kujutatud kohaselt.

Avamistegevused - skeemteksti avamisega seotud tegevused. Põhiülesanne on skeemteksti importimine süsteemi. Siia kuuluvad ka skeemteksti avamisele



Joonis 2.3: Töö skeemtekstiga.

eelnevad tegevused, nagu failide asukohtade (kataloogiteed, failide nimed) kindlaks tegemine, varukoopia loomine jms.

Toimetamine – skeemteksti toimetamine, toimetamise funktsioonide rakendamine.

Salvestustegevused - skeemteksti salvestamisele eelnevaid ja järgnevad tegevused. Näiteks tehakse eelnevalt kindlaks failide salvestuskohad (kataloogiteed, failide nimed), salvestatakse jooksev skeemtekst. Pärast võib toimuda skeemteksti formaadi teisendamine ja salvestamine ning tulemuse töötlemine.

Tüüpilised avamistegevused:

Import (või *Read Text*) – jooksva skeemteksti avamine
Export ... – varukoopia loomine

Tüüpilised salvestustegevused:

Export ... (või *Write Text*) – jooksva skeemteksti salvestamine
Textualize... – skeemtekst baaskeelseks tekstiks
Kompileerimine vm. töötlemine
Test-käivitus

Toimetatavad failid kujutavad endast tavaliselt skeem-modelleeritud arvutiteksti [2]. Toome veel mõned näited töö käigust erinevat tüüpi failidega süsteemis *Amadeus*.

Java failid

1. Skeemkujul *Java* klassi importimine süsteemi (*File+Import*).
2. Skeemteksti toimetamine süsteemi toimetamise funktsioonide abil.
3. Salvestamine:
   (a) Eksportida skeemkujul *Java* klass (*File+Export plain-HTML*).
   (b) Teisendada skeemtekst *Java* tekstiks (*File+Textualize*), tulemus kuvatakse uues aknas.
   (c) Teisendatud tekst kirjutada *.java* failiks (*File+Write text*).
   (d) Kompileerida *.java* fail.
   (e) Test-käivitus.

LaTeX failid

1. Skeemkujul *LaTeX* klassi importimine süsteemi (*File+Import*).
2. Skeemteksti toimetamine süsteemi toimetamise funktsioonide abil.
3. Salvestamine:
   (a) Eksportida skeemkujul *LaTeX* tekst (*File+Export plain-HTML*).
   (b) Teisendada skeemtekst *LaTeX* tekstiks (*File+Textualize*), tulemus kuvatakse uues aknas.
   (c) Teisendatud tekst kirjutada *.tex* failiks (*File+Write text*).
   (d) LaTeX töötlus – *latex Peafail.tex*, kus Peafail.tex kaasab osana (*input*) vaadeldava *.tex* faili.

2.1.3 Arendustöö projekti näide

Olgu näiteks üliõpilasel vaja teha semestritöö, mille raames tuleb välja töötada teatud lahendus ja lisada see mingisse suuremasse *Java* programmisüsteemi. Loomulikult peab ta lahenduse idee ja teostuse kirjelduse esitama ka semestritöö tekstina (mille vormiks olgu nt. LaTeX). Märgime, et semestritöö sooritamine on tegelikult just projekt selle üldises tähenduses (vt. epigraaf lk. 6). Semestritöö kui planeeritud tegevuse tulemuseks on

- lahenduse algoritmid,
- väljatöötatud *Java* klassid,
- semestritöö tekst (koos algoritmide skeemidega).

Vaatleme sellise arendustöö läbiviimist süsteemi *Amadeus* toel, esialgu ilma süsteemi *Amadeus* projekti kasutamiseta.

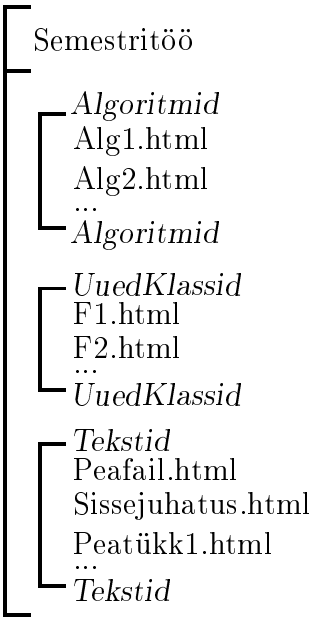
Failid

Iga algoritm sisestatakse/toimetatakse kui skeemtekst; salvestatakse mingisse faili (nt. Alg*n*.html) algoritmide kaustas nimega "Algoritmid". Iga loodud *Java* klass sisestatakse/toimetatakse kui skeemtekst baaskeelega *Java*; salvestatakse mingisse faili (nt. F*n*.html) dokumentatsiooni kaustas ("UuedKlassid"). Semestritöö teksti osad sisestatakse/toimetatakse kui skeemtekst baaskeelega LaTeX; salvestatakse mingisse faili (nt. on osadeks Peafail.html, Sissejuhatus.html, Peatükk1.html, ...) tekstide kaustas ("Tekstid"). Vastav kaustastruktuur on kujutatud joonisel 2.4.

Algoritmide töötlemine

Algoritm Alg*n*.html avatakse süsteemi *Amadeus* aknas, toimetatakse ja salvestatakse. LaTeX trükikuju saamiseks tehakse menüükäsk *Tools*+*PrepareTex*; tekib uus aken, milles Alg*n*.html skeemtekstis on LaTeX erisümbolid sobivalt asendatud, näiteks iga sümbol '\' on asendatud sõnaga '$\backslash$' [1]. Uues

[1] Selline TeX-valmistuse rakendamine eeldab, et skeemteksti Alg*n*.html baaskeleks on seatud *Java*.



Joonis 2.4: Semestritöö üldine kaustastruktuur.

aknas olev skeemtekst salvestatakse TEX-vormis, menüükäsuga *File+Print plain-TEX* faili Alg*n*.tex. Salvestuskohaks võiks olla kausta "Tekstid" alamkaust "Latex".

Klasside töötlemine

Skeemkujul klass F*n*.html avatakse süsteemi *Amadeus* aknas, toimetatakse ja salvestatakse. Vastava *Java* lähtekoodi saamiseks tehakse menüükäsk *Tools+Textualize*, tekib uus aken, milles kuvatakse klassi F*n* *Java*-tekst. Viimane salvestatakse menüükäsuga *File+Write tekst* faili F*n*.java. Salvestamise kohaks on semestritöös täiustatava suurema süsteemi *Java* lähtekoodi kaust, nt. D:\BigSystem\Source. Seejärel tuleb *Java* lähtekood F*n*.java kompileerida, nt.

```
C:\"Program Files"\j2sdk1.4.2_02\bin\javac
                        -sourcepath "D:\BigSystem\Source"
                        -classpath "D:\BigSystem\Source\class"
```

`-d "D:\BigSystem\Source\class" F`*n*`.java`

Lõpuks käivitada testimiseks kogu suurem süsteem, nt.

`C:\"Program Files"\j2sdk1.4.2_02\bin\java BigSystem`.

Tekstide töötlemine

Skeemkujul tekstiosa, olgu selleks nt. Peatükk1.html, avatakse süsteemi *Amadeus* aknas, toimetatakse ja salvestatakse. Vastava LaTeX teksti saamiseks tehakse *Tools*+*Textualize*, tuleb uus aken, milles kuvatakse vastav LaTeX tekst. Seejärel töödeldakse (`latex -halt-on-error`) Peafail.tex, mis kaasab kõik teksti osad, sh. osa Peatükk.tex ja ka TeX kujul algoritmid Alg*n*.tex.

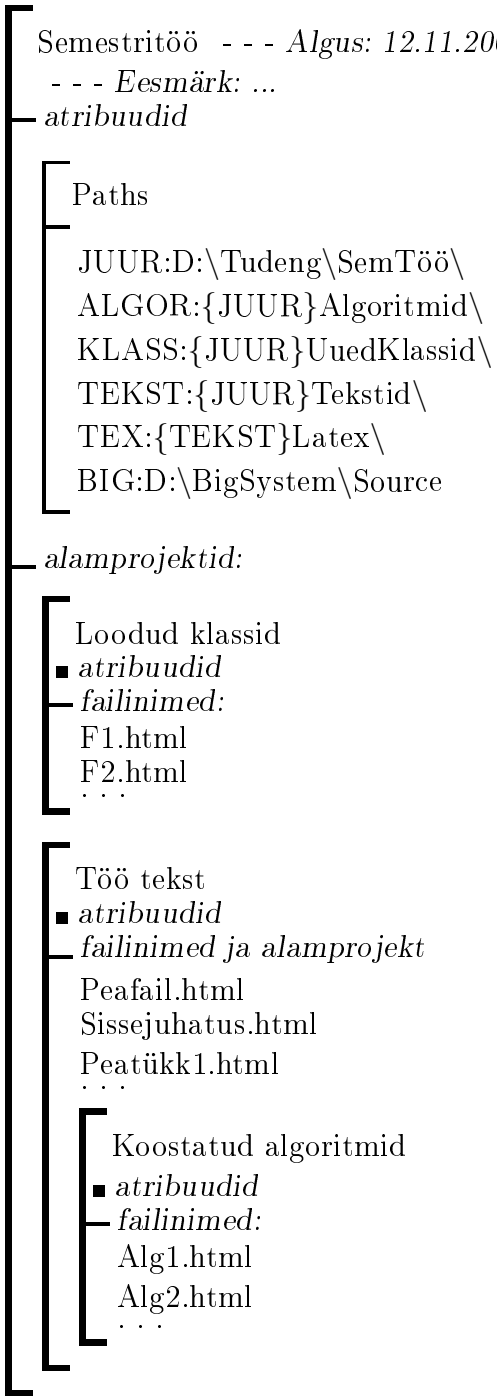
Nagu näha, on peale failide toimetamise/salvestamise veel vaja sooritada palju tegevusi, mis nö. käsitsi tegemisel oleksid tüütult rutiinsed. Pealegi peab kuskil eraldi meeles pidama kausta struktuuri (kus mingit tüüp fail paikneb) ja hulganisti tegevuste (käsuridade) üksikasju.

Ette rutates võib öelda, et kogu ülaltoodud kirjelduse saab esitada süsteemi *Amadeus* projektina. Seejuures on viimane isegi "täidetav": kõik ettenähtud tegevused rakenduvad automaatselt (vastava töödeldava üksuse – algoritm, klass, tekst – aknas menüükäsu *File*+*Save* toimel).

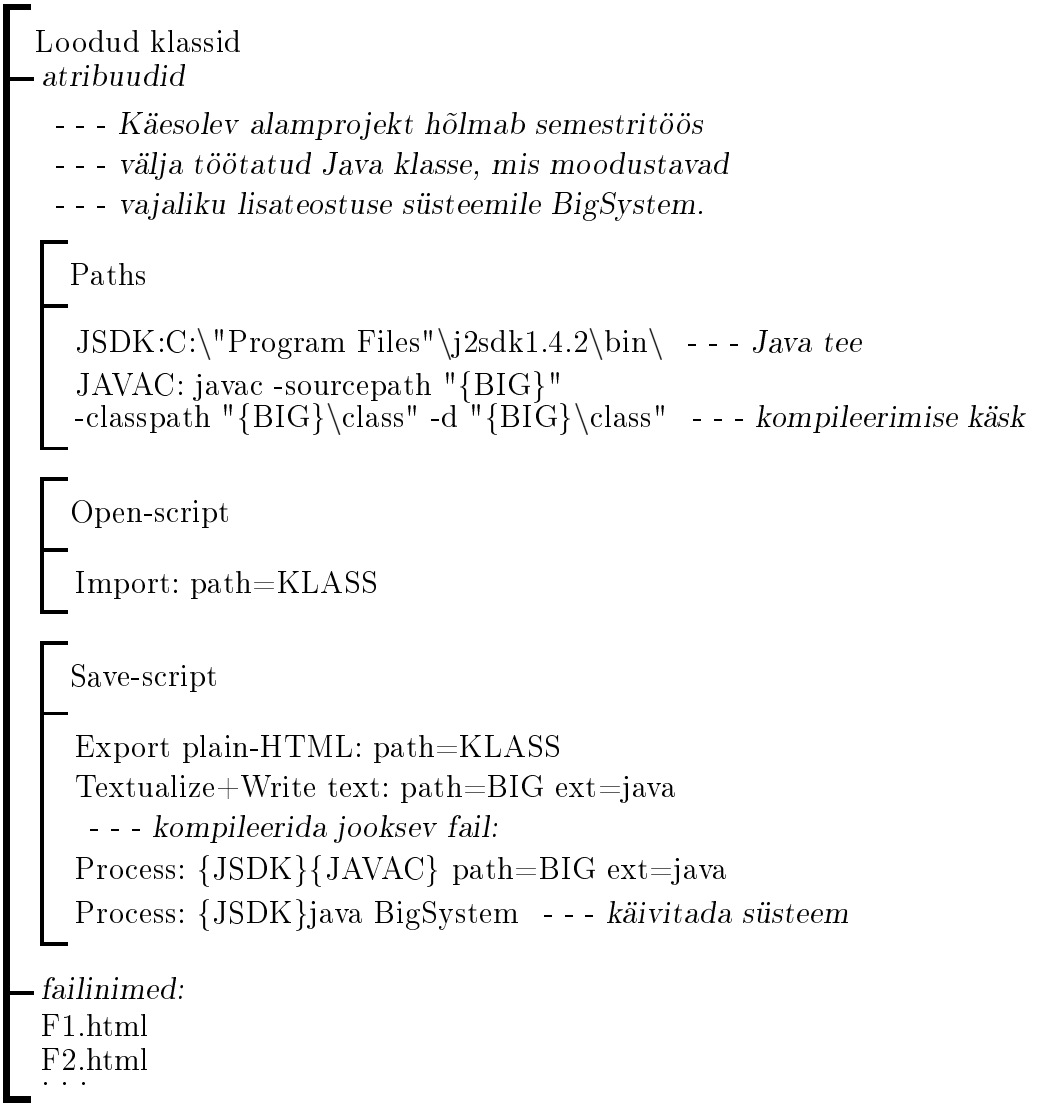
Näitena vaadeldud semestritöö süsteemi *Amadeus* projektina on esitatud joonistel 2.5, 2.6, 2.7. Ruumi kokkuhoiu mõttes ei ole semestritöö kui projekti kogu kirjeldavat osa kuvatud. Väikese erinevusena ülalkirjeldatust on järgnevas süsteemi *Amadeus* projektis ette nähtud ka algoritmi kohene töötlus (`latex Peafail.tex`), kuna algoritmid moodustatavad semestritöö teksti Peafailist kaasatava osa.

2.2 *Amadeus* projekti määratlus

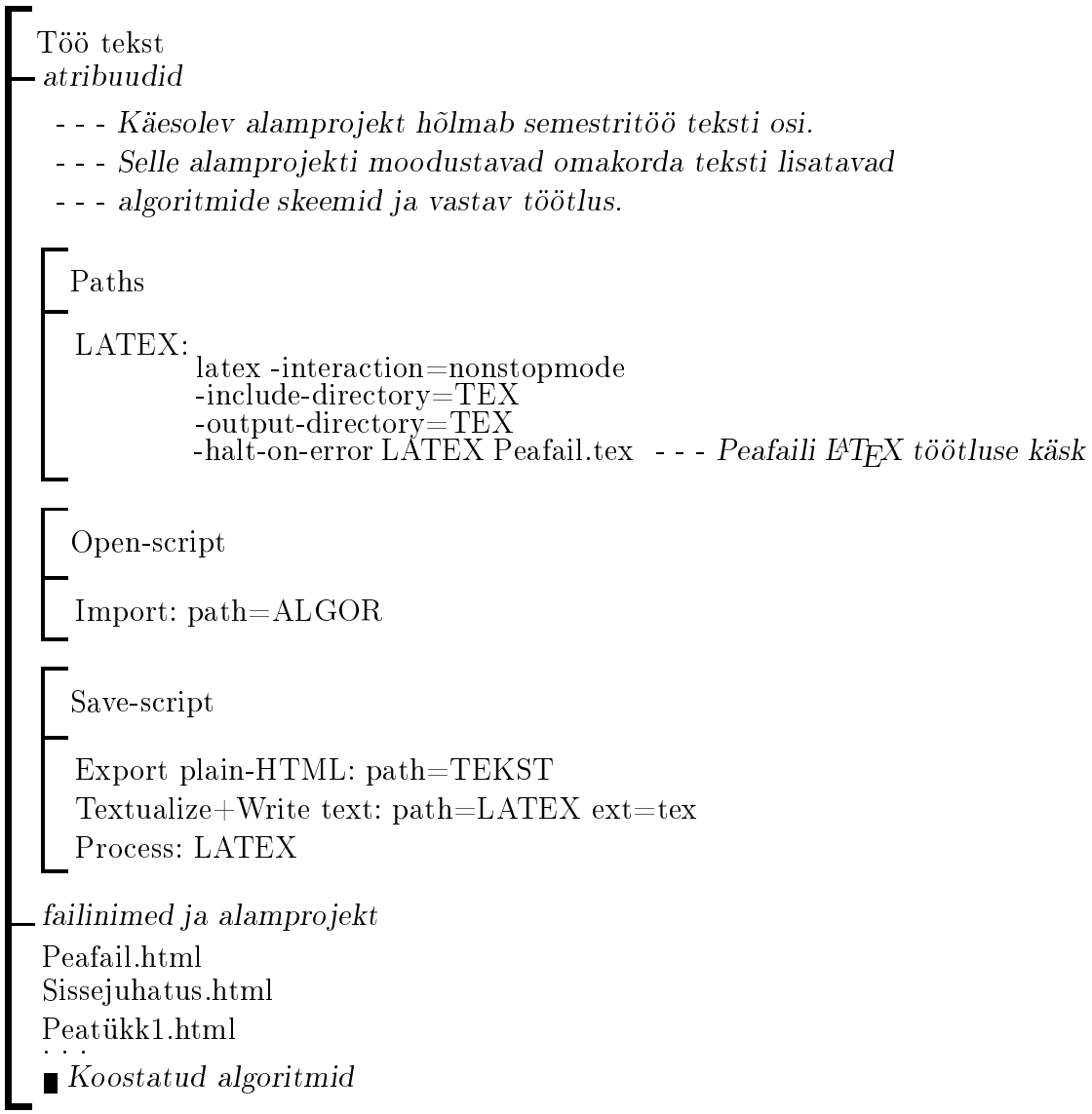
Süsteemi *Amadeus* aktiivsel kasutajal kujunevad välja kindlad failide avamise ja salvestamisega seonduvad tegevused (tegevuste jadad), mida ta regulaarselt kordab failidega töötamisel. Korduvate tegevuste arv kasvab proportsionaalselt failide arvuga. Lisaks sellele ähvardab väiksemagi eksimuse puhul veel salvestamata andmete kaotsiminek. Näiteks käesoleva töö alusversioonis suletakse raam ilma mingi täiendava hoiatuseta. Kuid iga rutiinset tööd



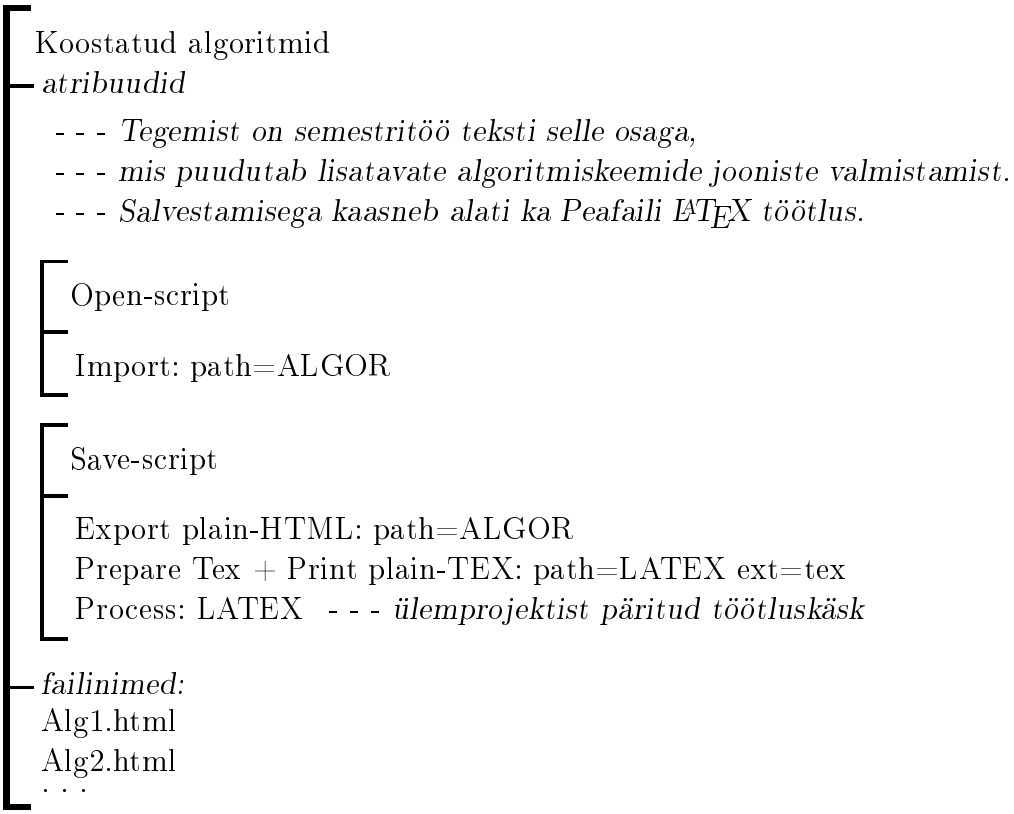
Joonis 2.5: Semestritöö projekt: ülevaade.



Joonis 2.6: Semestritöö projekt: alamprojekt 'Loodud klassid'.



Joonis 2.7: Semestritöö projekt: alamprojekt 'Töö tekst'.



Joonis 2.8: Semestritöö projekt: alamprojekt 'Koostatud algoritmid'.

tuleks püüda (ja ka saab) automatiseerida

- säästmaks kasutaja tööaega,
- vältimaks kasutaja hajameelsusega seotud vigu ja raskeid eksimusi.

Kui toimetamise funktsioonid on iga faili jaoks erinevad, siis töötlemise funktsioonid aga võivad olla ühesugused kõigile mingi tunnuse järgi grupeeritud failidele. Näiteks selle järgi, milline on failide tüüp, failide asukoht või failide kasutamise viis (ka otstarve). Järgnevas projekti määratluses lähtumegi sellest, et projekt rühmitab failinimesid, kus rühmas on iga failinimega seotud ühesugused toimetamise eel- ja järeltegevuste avamis- ja salvestustegevuste sooritamiseks vajalikud andmed ehk atribuudid.

2.2.1 Projekt ja projektikirjeldus

Käesolevas vaadeldav hierarhilise projektikirjelduse mõiste leiab käsitlemist ka artiklis [1].

Projekt on paaride (a, f) hulk, kus f on failinimi ja a on sellega seotud atribuudid. Projektis esinevad failinimed on paarikaupa erinevad, st. korduvaid failinimesid ei ole. Tegelikus töös esinevad enamasti projektid, milles paljude failide atribuudid on samad.

Projektikirjeldus on projekti üleskirjutus, mis võimaldab kokku võtta samade atribuutidega failid.

Lihtsamal juhul koosneb projektikirjeldus atribuudiosast ja failide nimekirjast. Näiteks projektikirjeldus

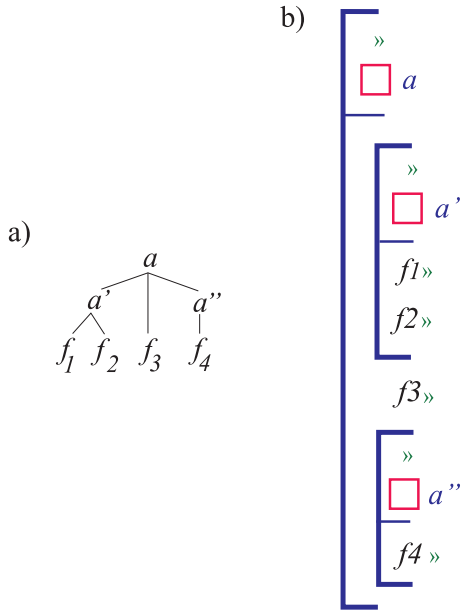
$$(a;\ f_1\ ,f_2,\ \ldots\ ,\ f_n)$$

sätestab, et faili(nime)de f_1, f_2, ... , f_n atribuutideks on a.

Üldjuhul koosneb projektikirjeldus atribuudiosast ja failinimede ja/või projektide loetelust:

$$(a;\ r_1\ ,r_2,\ \ldots\ ,\ r_n),$$

kus r_i on failinimi või (alam)projekti kirjeldus ($i = 1,\ 2,\ \ldots\ ,n$). Seega on projektikirjeldus hierarhiline struktuur (projektipuu), mille juureks on atribuudiosa ja milles lehtedeks on failinimed, ning iga vahetipp on omakorda projektipuu juureks. Näiteks projektikirjelduse $(a;\ (a';\ f_1,\ f_2),\ f_3,\ (a'';\ f_4))$ hierarhiline struktuur on kujutatud joonisel 2.9.



Joonis 2.9: Projektikirjelduse hierarhia puuna(a) ja skeemtekstina(b).

Projektikirjelduses lehena esineva failinime atribuudid pärinevad atribuudiosadest üles-teel sellest lehest kuni kogu projektipuu juureni. Näiteks projektikirjelduse $(a;\ (a';\ f_1,\ f_2),\ f_3,\ (a'';\ f_4))$ korral

- failinime f_3 atribuudid on määratud atribuudiosaga a;
- failinimede f_1 ja f_2 atribuudid pärinevad atribuudiosadest a ja a';
- failinime f_4 atribuudid pärinevad atribuudiosadest a ja a''.

Atribuutide pärilusmehhanism on täpsemalt kirjeldatud jaotises 2.2.4. Edasises loeme terminid *projekt* ja *projektikirjeldus* sünonüümideks.

* * *

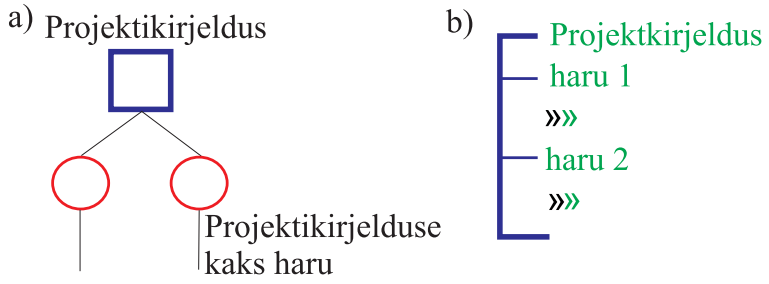
Süsteemis *Amadeus* loetletakse projektikirjelduses failide lihtnimed, kataloogiteedega prefikseerimata kujul. See tähendab, et projektis on tegemist nö. virtuaalsete failide loeteluga, mistõttu failide loetelu projektis on konkreetsest failisüsteemist sõltumatu. Tegelikus töötluses kasutatavad

(füüsilised) failid määratletakse sel teel, et failinimed prefikseeritakse dünaamiliselt kataloogiteedega. Viimased aga antakse projektikirjelduses eraldi, nimelt failinimede atribuutidena. Näiteks üleminekul mingisse teise failisüsteemi, kus on teised kataloogiteed (kuid projektifailide otsene kaustastruktuur sama) tuleb projektis muuta ainult atribuutide osa, failide nimekirjad jäävad aga muutumatuks.

2.2.2 Projektikirjelduse skeemmudel

Projektikirjelduse esitamiseks ei ole kasutusele võetud mingisugust spetsiaalset keelt, vaid projektikirjeldus antakse standardses skeemteksti vormis. Projektikirjelduse skeemmudel sätestab, milliseid skeemelemente mis otstarbeks tuleb kasutada.

Projektikirjeldus esitatakse nimelt moodulskeemina, millel on üks elementaarpäis ja kaks haru. Skemaatiline näide projektikirjeldusest skeempuuna on esitatud joonisel 2.10.

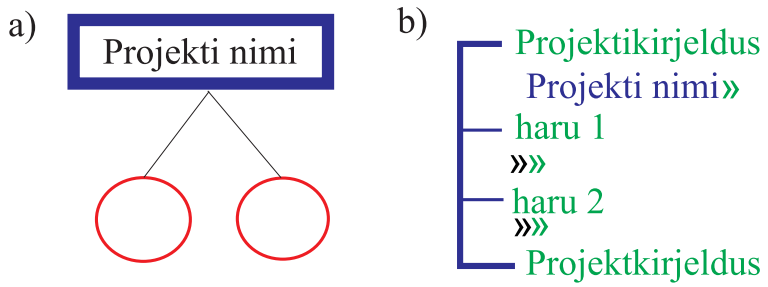


Joonis 2.10: Projektikirjeldus skeempuuna(a) ja skeemtekstina (b).

Skeemteksti, mis sisaldab vähemalt ühte projektikirjeldust, nimetatakse ka *portfelliks*. Portfellis peab iga moodulskeem olema projektikirjeldus. Paneme tähele, et iga projektikirjeldus on portfell.

Projektikirjelduse elementaarpäise tekstiks on projekti nimi. Joonisel 2.11 elementaarpäise tekst kirjutatud tipu sisse. Projekti nimi ei saa olla tühisõne. Ühes *portfellis* ei tohi olla kahte sama nimega projekti.

Projektikirjelduse esimeses harus asub 0 või enam projekti atribuudirühma. Projekti *atribuudirühm* on ühe lihttüüpi elementaarpäisega üheharuline



Joonis 2.11: Projektikirjelduse moodulskeem elementaarpäisega.

lihtskeem. Joonisel 2.12 on näidatud projektikirjelduse *HELP* esimene haru. Haru sisaldab kolm atribuudirühma: kataloogiteede lihtskeem (atribuudirühm nimega *Paths*), avamistegevuste lihtskeemi (*Open-script*) ja salvestamistegevuste lihtskeemi (*Save-script*).

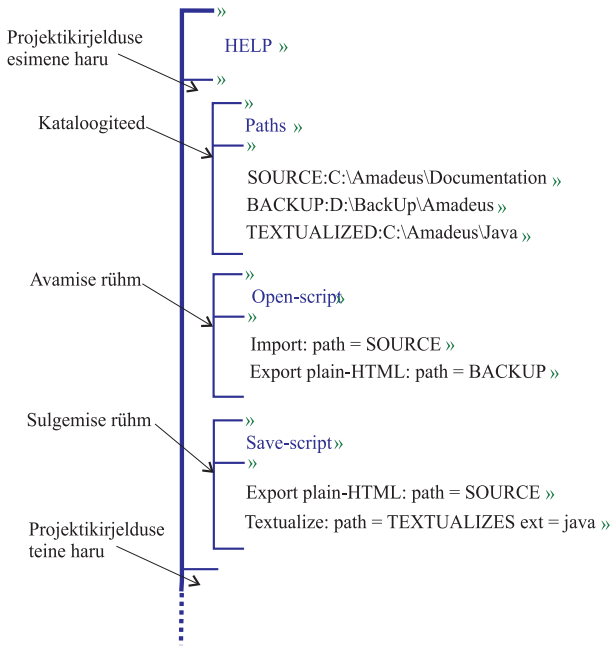
Atribuudirühma koondatakse teatavas mõttes kokkukuuluvate (ühise otstarbe järgi grupeeritud) atribuudid[2]. Atribuudirühm kujutatakse skeemtekstis päisega lihtskeemina: igale atribuudirühmale vastab üheharuline ühe elementaarpäisega lihtskeem. Atribuudirühma nimeks on lihtskeemi päise tekst. Iga atribuut esitatakse lihttüüpi primitiivi tekstina. Joonisel 2.12 atribuudi rühmas *Paths* on defineeritud kolm atribuuti: SOURCE, BACKUP ja TEXTUALIZED.

Ühes projektikirjelduses võib olla kuitahes palju erinevate nimedega atribuudirühmi. Tõsi küll, käesolevas versioonis töödeldakse vaid atribuudirühmi nimedega *Paths*, *Save-script* ja *Open-script*.

Projektikirjelduse teise haruga antakse projektifailide nimed (iga nimi on ühe primitiivliikme tekstiks) ja alamprojektide kirjeldused. Joonisel 2.9(a) oleva projektipuu esitus skeemtekstina on kujutatud joonisel 2.9(b).

Nii portfellis kui ka igas projektikirjelduses võib üsnagi vabalt kasutada mitmesuguseid täiendavaid skeeme (küll mitte moodulskeeme) ja muid skeemielemente. Näiteks on joonisel 2.13 vasakus skeemtekstis kasutatud kolme grupeerivat lihtskeemi ja ühte topeltskeemi. Need vahendid on kommenteeriva iseloomuga ega muuda projektikirjelduse (allpool esitatud) sisulist tähendust. Joonisel 2.13 on mõlemad skeemtekstid samaväärsed projektikir-

[2]Atribuudi mõistet vaadeldakse lähedamalt jaotises 2.2.3).



Joonis 2.12: Projektikirjelduse atribuudid.

jeldused ja kirjeldavad projekti
(*projekti nr.* 1 *atribuudid*; $fail1.html$, $fail2.html$,
(*projekti nr.* 2 *atribuudid*; $fail3.html$, $fail4.html$),
$fail5.html$, $fail6.html$)
).

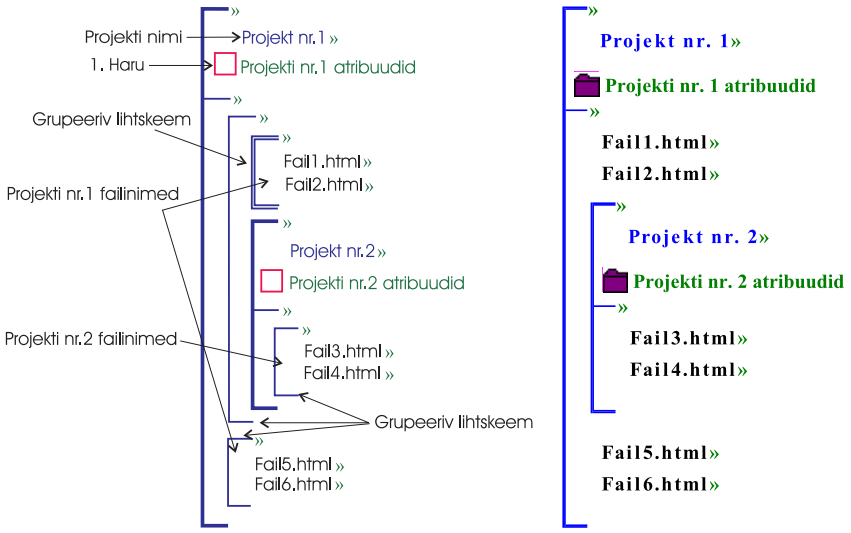
Kõik skeemelementide kommentaarid on projektis vabalt kasutatavad ega muuda mingilgi määral projektikirjelduse tähendust.

2.2.3 Atribuudid

Atribuudid kirjeldatakse atribuudirühmades.

Atribuudid *Path* rühmas

Atribuute *Path* rühmas nimetatakse tee-atribuutideks. *Path* atribuudirühmas on atribuudi süntaks järgmine:



Joonis 2.13: Grupeeritud osadega projektikirjeldus (vasakul), sama projektikirjeldus degrupeerituna (paremal).

S <märgend> S':'S <väärtus> S

kus S on 0 või enam tühikut, <märgend> on mittetühi kooloni- ja tühikuvaba sõne; <väärtus> on üks järgmistest:

- kataloogitee
- kataloogitee osa
- faili nimi
- käsurea tekst või selle osa

Näiteks, tee-atribuudis

SOURCE: D:\Amadeus3\Documentation

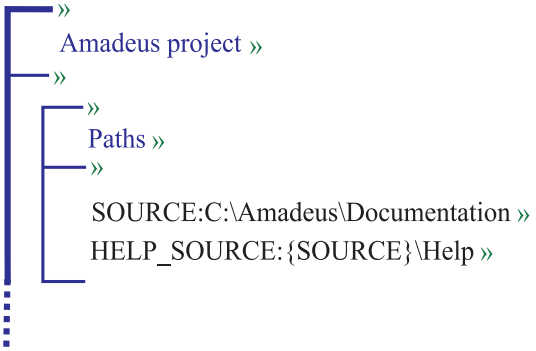
on järgmised osad:

SOURCE – atribuudi märgend, ehk nimi,
D:\Amadeus3\Documentation – atribuudi väärtus.

Path rühmas on atribuutide märgendid projektikirjelduse koostaja poolt vabalt valitavad. Märgend identifitseerib seda atribuuti ühe projektikirjelduse sees.

Path rühmas defineeritud atribuutide väärtusi saab kasutada nende märgendite abil teiste atribuutide väärtuste defineerimisel kahel teel:

looksulgude vahel – {*atribuudi_ märgend*}, kus *atribuudi_ märgend* peab olema defineeritud kas samas projektikirjelduses või pärinema mõnest ülemprojektikirjeldusest. Looksulud koos märgendiga asendatakse vastava märgendi väärtusega. Näiteks joonisel 2.14 atribuudi HELP_FILES täisväärtus on C:\Amadeus\Documentation\Help.



Joonis 2.14: Atribuut teise atribuudi väärtuse osa.

Looksulgudes märgendeid saab kasutada nii *Path* rühmas kui ka tegevuste rühmades atribuutide defineerimisel. Viimastes küll ainult Process funktsiooni defineerimisel (vt. lk. 44).

parameetri väärtusena – path = *atribuudi_ märgend* või ext = *atribuudi_ märgend*. Niisugusel viisil saab märgendeid *Path* rühmast kasutada ainult tegevuste rühmades (*Open-script* või *Save-script*) ja märgendi väärtust kasutatakse sõltuvalt parameetrist. Parameetri *path* abil määratakse kas töödeldava faili asukoht või töödeldava faili nimi. Kui vastava märgendi väärtus on kataloogitee, siis seda teed kasutatakse avatavale või salvestatavale failinimele uue tee määramiseks. Kui vastava märgendi väärtus ei ole mingi kataloogitee vaid failinimi, siis eeldatakse, et märgendi väärtus määrab täielikult töödeldava faili nime. Parameeter *ext* määrab töödeldava faili uue laiendi.

Atribuudid tegevuste rühmades

Süsteemi *Amadeus* uues versioonis on projektifaili akna *File* menüüse lisatud kahe üldfunktsiooni menüüelemendid *Open* ja *Save*. Atribuutrühmad *Open-script* ja *Save-script* kirjeldavad tegevuste jadasid, mis rakenduvad vastavate menüüvalikute *File+Open* ja *File+Save* valimise korral. Normaalselt sisaldab *Open-script* rühm skeemteksti toimetamisele eelnevaid tegevusi (sh. skeemteksti avamine), *Save-script* aga skeemteksti toimetamisele järgnevaid tegevusi (sh. skeemteksti salvestamine).

Kõik *Amadeus* funktsioonid saab jaotada kolme tüüpi. Järgnevas on funktsioonide nimed võetud vastavalt inglisekeelsest menüüst.

- Funktsioonid, mis nõuavad sisendiks ainult faili nime. Seda tüüpi on kõik avamise (menüü *File*) funktsioonid - *Import*, *Insert* ja *Read text*. Niisuguste funktsioonide tulemuseks on uus *Amadeus* aken, kus avatud faili sisu on esitatud skeemtekstina.

- Funktsioonid, mis kasutavad sisendparameetrina ainult jooksvat skeemteksti (raamis olevat skeemteksti). Seda tüüpi on kõik teisendusfunktsioonid (menüü *Tools*), nt. *Sketchify*, *Reduce*, *Normalize* jm. Nende funktsioonide tulemuseks on töödeldud skeemtekst, mida näidatakse kas samas või siis uues raamis.

- Funktsioonid, mis nõuavad töötamiseks nii faili nime kui ka jooksvat skeemteksti. Näiteks (menüü *File*) kõik eksportfunktsioonid ja funktsioonid *Write text*, *Print plain-TEX* ja *Print PostScript*. Need funktsioonid väljastavad raamis oleva skeemteksti antud nimega faili.

Nii *Save-script* kui ka *Open-script* tegevuste atribuutides saab kasutada kõiki menüüde (*File* ja *Tools*) funktsioone, kuid *Open-script* tegevuse puhul tuleb jälgida, et failile oleks kõigepealt rakendatud avamise funktsioon, mis konverteerib faili teksti skeemtekstiks, ja seejärel rakendada funktsioone, mis kasutavad skeemteksti.

Atribuudirühmad *Open-script* ja *Save-script* sisaldavad *tegevus-atribuute*, mille süntaks on järgmine:

S <käsk> S':'S <parameetrid> S

kus S on 0 või enam tühikut. Käsk koosneb ühest või mitmest (ingliskeelsest) menüüvaliku tekstist (funktsiooni nimetus, milles tähtede tõst ei oma tähtsust), kusjuures eraldajaks on S'+'S. Parameeter esitatakse kujul:

<parameetri nimi> S'='S <parameetri väärtus>

Parameetrite eraldajaks on üks või mitu tühikut. Parameetrite nimed on käsuspetsiifilised, käesolevas versioonis kasutatakse ainult kahte parameetri nime – *path* ja *ext*.

Kui tegevus-atribuut eeldab, et atribuudi käsk rakendatakse mingi etteantud faili (nn. töödeldava faili) nimele, siis parameetrite *path* ja *ext* eesmärgiks on muuta töödeldava faili nimi "sobivaks". Parameeter *path* määrab töödeldava faili uue tee failisüsteemis ja parameeter *ext* määrab uue faililaiendi: väärtus antakse mingi selles projektikirjelduses kehtiva tee-atribuudi märgendi näol.

Process funktsioon

Lisaks menüükäskudele saab tegevus-atribuudiks olla ka *Process* funktsioon.

Süsteemi *Amadeus* erifunktsioon *Process* saab argumendiks käsurealt käivitava käsuteksti, käivitab selle ja väljastab vead (kui niisugused tulevad) *Amadeus* uues raamis. *Process* käsu defineerimisel projektikirjelduses ongi parameetriteks käsurea tekst, kuid käsurea teksti defineerimisel saab kasutada *Path* rühma atribuute, kas looksulgude või parameetrite *path* ja *ext* abil. *Process* funktsiooni kirjeldav tegevus-atribuut esitakse eraldi primitiivina.

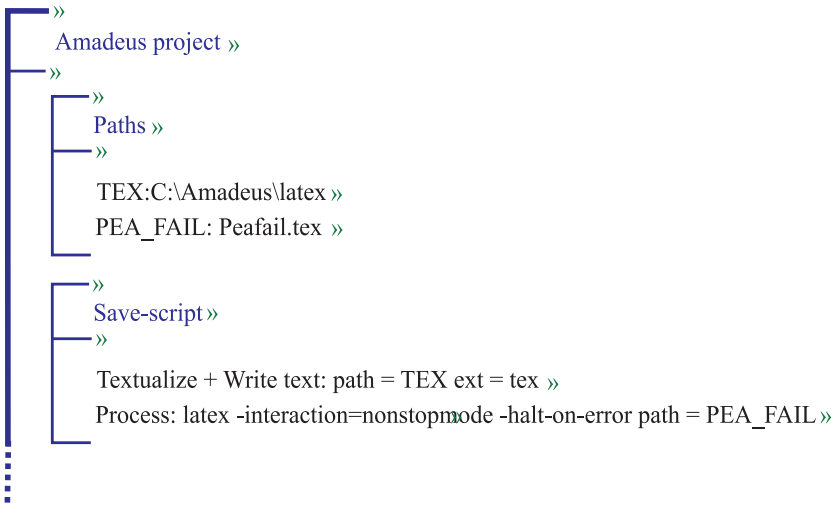
Process käsu parameetrite *path* ja/või *ext* väärtuste abil moodustatakse faili nimi.

Looksulud koos tee-atribuudi märgendiga asendatakse selle atribuudi väärtusega. Joonisel 2.15 (viimases reas oleva) *Process* käsu saaks defineerida ka kujul:

```
Process: latex -interaction=nonstopmode -halt-on-error {PEA_FAIL}
```

Mõlemal juhul on käsurea terviktekstiks:

```
latex -interaction=nonstopmode -halt-on-error Peafail.tex
```



Joonis 2.15: *Process* käsu kasutamine.

Kui looksulgudes kasutatud märgendi jaoks ei ole võimalik mingi põhjusel atribuuti leida, siis märgendi (koos teda ümbritsevate looksulgudega) asendust ei toimu.

Keerulisem *Process* funktsiooni kasutamise näide on joonisel 2.16. Eeldades, et töödeldava faili nimi on *AmFrame*, oleks defineeritud *Process* funktsiooni käsurea terviktekst:

```
C:\j2sdk1.4.2_02\bin\javac -sourcepath "C:\Amadeus\java" -classpath
 C:\Amadeus\class -d C:\Amadeus\class C:\Amadeus\java\AmFrame.java
```

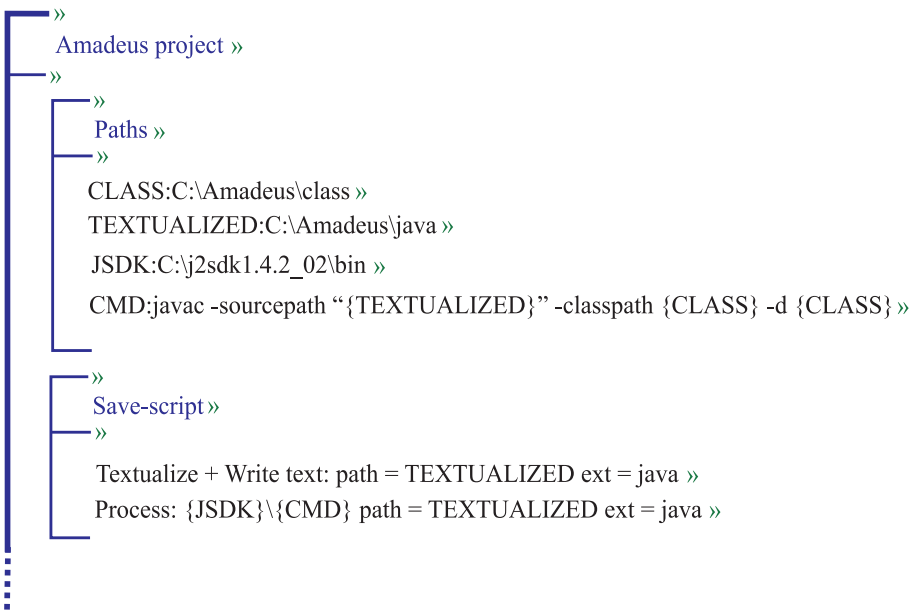
Process käsu koostamise korrektsus on täielikult kasutaja vastutusel.

Näiteid

Järgnevas on antud mõned tegevus-atribuutide näited koos selgitustega. Esimesed kaks näidet on *Open-script* rühma atribuutide, viimased kaks aga *Save-script* rühma atribuutide kohta. Eeldatakse, et kõik tegevus-atribuudid rakendatakse projektifailile nimega *f.txt*.

Näide 1

Read text + Sketchify: path = SOURCE



Joonis 2.16: *Process* käsu kasutamine (keerulisem variant).

Open-script tegevus-atribuut. Avada tekst fail *f.txt*, mis asub kohas SOURCE ja sellele rakendada skeemistamise (*Sketchify*) funktsiooni.

Näide 2

Import: path SOURCE
Export plain-HTML: path = BACKUP

Open-script tegevus-atribuudid. Importida fail *f.txt* mis asub kohas SOURCE ja teha temast varukoopia käsu Export plain-HTML abil kohta BACKUP.

Näide 3

Textualize + Write text: path=TEXTUALIZED ext=java

Save-script tegevus-atribuut. Tekstualiseerida skeemtekst *f.txt* ja seejärel salvestada tulemus tee-atribuudi TEXTUALIZED abil määratud kataloogi; salvestamisel määrata failinime laiendiks *java*. Tulemusfail *f.java* kirjutatakse kausta TEXTUALIZED.

Näide 4

Process: javac path=TEXTUALIZED ext= java

Save-script tegevus-atribuut. Selle tegevus-atribuudi täitmise tulemusena kompileeritakse kataloogis TEXTUALIZED asuv *Java* fail *f.java*. Eeldatakse, et kataloogis TEXTUALIZED leidub fail nimega *f.java*.

2.2.4 Atribuutide pärinemine

Alam projektikirjeldustes atribuut saab olla üledefineeritud või laiendatud päriluse teel. Viimasel juhul algab atribuudi väärtus reserveeritud sümbolite kombinatsiooniga – "."+*kataloogitee eraldaja*. Pärimine toimub kahel tasemel. Esiteks atribuudirühma tasemel, teiseks atribuudi tasemel. Atribuudi tasemel pärimine toimub ainult tee-atribuutide jaoks.

[I] Kui projektikirjelduse esimeses harus puudub mingi atribuudirühm, siis kasutakse atribuudi rühma lähimast ülemprojektikirjeldustest, kus sama nimega rühm on olemas.
St. projektikirjelduses p võime kasutada ülemprojektikirjelduse atribuudirühma r atribuute, kui projektikirjelduses p atribuudirühm nimega r puudub.
Teiste sõnadega projektikirjelduses võime kasutada kõiki ülemprojektide atribuudirühmi, mida ei kaeta üle selles projektikirjelduses (samaniimelise atribuudirühmas).

[II] (a) Kui projektikirjelduse esimeses harus *Open-script* ja *Save-script* tegevuste rühmades kasutatav tee-atribuut puudub sama projektikirjelduse tee-atribuutide rühmas siis otsitakse sama nimega atribuut ülemprojektikirjeldustest.

(b) Kui projektikirjelduse esimeses harus *Open-script* ja *Save-script* tegevuste rühmades kasutatav tee-atribuudi väärtus on formaadis

<*punkt*><*kataloogi tee eraldaja*><*tee*>

siis atribuudi väärtuse leidmiseks rakendatakse pärimise mehhanismi.

Tee-atribuudi pärimise mehhanism

Pärimise mehhanism laiendab tee-atribuudi väärtuse sama nimega tee-atribuudi väärtusega ülemprojektikirjeldustest. Pärimise mehhanismi töö lõpeb, kui tee-atribuudi leitud väärtus on faili süsteemi täistee või kui kõik ülemprojektikirjeldused on läbi vaadatud. Pärimise mehhanism eeldab, et koostatakse kataloogitee (või faili nimi koos teega). Seega süsteem hoolitseb, et liituvate atribuutide väärtuste vahele jääks kindlasti täpselt üks kataloogi tee eraldaja.

Näide

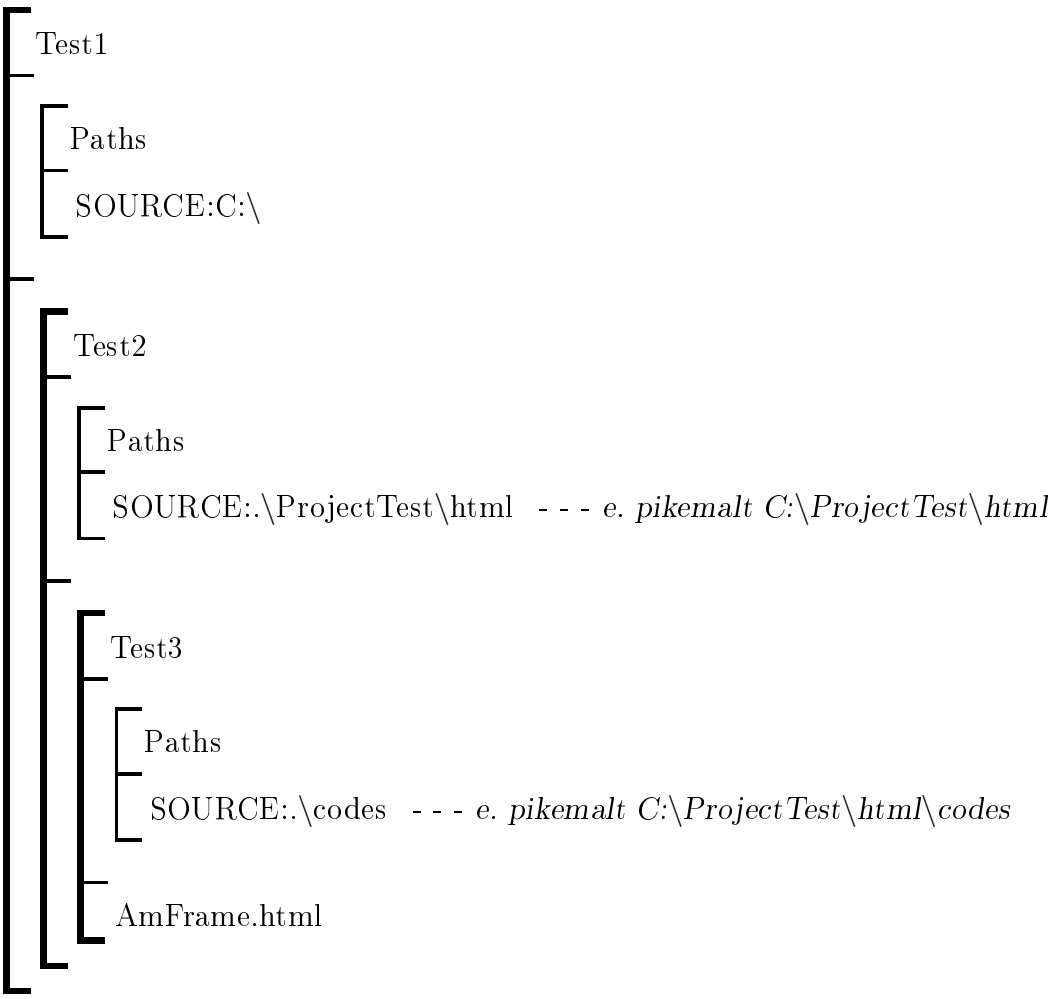
Olgu on antud kolm projektikirjeldust, kus *Test2* on projekti *Test1* alamprojektikirjeldus ja *Test3* on projekti *Test2* alamprojektikirjeldus. Vastav skeemtekst paikneb joonisel 2.17. Tee-atribuudi SOURCE täisväärtus iga projektikirjelduse jaoks on näidatud tabelis 2.1.

Projekti nimi	Tee-atribuudi SOURCE defineerimine	Tee-atribuudi SOURCE pärimise mehhanismiga saadud väärtus
Test1	SOURCE:C:\	C:\
Test2	SOURCE:.\ProjectTest\html	C:\ProjectTest\html
Test3	SOURCE:.\codes	C:\ProjectTest\html\codes

Tabel 2.1: Tee-atribuudi SOURCE pärilus-väärtustamine.

Märkus: *Test1* ja *Test2* ei pea olema otsesed ülemprojektikirjeldused vastavalt *Test2* ja *Test3* jaoks. Näiteks *Test2* ja *Test1* vahel võivad olla vaheprojektikirjeldused, kui need ei sisalda tee-atribuuti SOURCE oma tee-atribuudi rühmas.

Tegevuste atribuudirühmas, milles ei ole ühtegi atribuuti, ei toimu mingeid lisategevusi. Sellist tühja atribuudirühma saab kasutada nt. samanimelise rühma ülemprojektist pärinemise keelamiseks.



Joonis 2.17: Atribuutide pärilus.

Peatükk 3

Projekti teostus süsteemis *Amadeus*

Magistritöös on välja töötatud süsteemi *Amadeus* uus versioon nimega *AmadeusJS*, lähtudes eelmisest versioonist *Amadeus-fRED*.

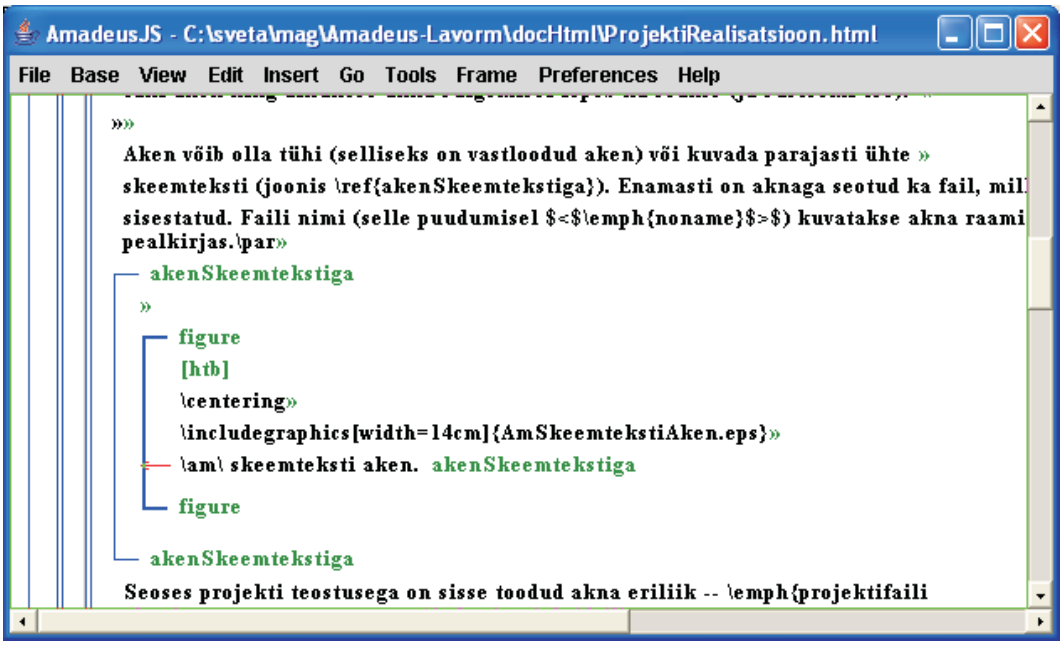
3.1 Projektorienteeritud kasutajavaade

Kõrvuti projekti mõistega on teostuse teiseks oluliseks lähtepunktiks täpsustatud kasutajavaade, mida süsteem peab täielikult toetama. Arusaadavalt peetakse siinkohal silmas kasutajat, kes töötab projekti(de)ga süsteemis *Amadeus*.

3.1.1 Tööruum ja aknad

Amadeus-seansi *tööruumi* moodustavad kõik parajasti olemasolevad *Amadeus*-raamid ehk aknad. Kasutaja saab luua aknaid (menüükäsuga *Frame+New*) ja neid sulgeda (kasutades süsteemseid vahendeid, nt. *Alt+F4*). Uusi aknaid tekib ka mitmete süsteemi *Amadeus* funktsioonide rakendamise tulemusena. Tööruumis on alati vähemalt üks aken, sest süsteemi käivitamisel luuakse kohe üks tühi aken ning ainukese akna sulgemisel lõpeb ka seanss (ja süsteemi töö).

Aken võib olla tühi (selliseks on vastloodud aken) või kuvada parajasti ühte skeemteksti (joonis 3.1). Enamasti on aknaga seotud ka fail, millest jooksev skeemtekst on sisestatud. Faili nimi (selle puudumisel <*noname*>) kuvatakse akna raami pealkirjas.



Joonis 3.1: *Amadeus* skeemteksti aken.

Seoses projekti teostusega on sisse toodud akna eriliik – *projektifaili aken*. Selle iseärasused, võrreldes ülalkirjeldatud aknaga, mida edaspidi nimetame *tavaliseks tööaknaks*, on järgmised.

- Projektifaili aken luuakse ainult seoses mingi kindla projektikirjeldusega ning on sellega permanentselt seotud. Akna paremas ülemises nurgas kuvatakse vastava projekti nimi (erijuhul *None*).
- Projektifaili aknas kuvatakse ainult mingit projektifaili.
- Projektifaili akna *File*-menüüs on aktiveeritud valikud *Open* ja *Save*, mis tavalise tööakna korral on deaktiveeritud. Keelatud on (projektiväliste) failide toomine/loomine projektiaknasse: deaktiveeritud on *File*-menüü valikud *New*, *Import*, *Input java* ja *Read text*.

Projektikirjeldusi käideldakse reeglina tavalistes tööakendes. Tõsi küll, põhimõtteliselt on see võimalik ka projektifaili akendes – juhul, kui projektikirjeldus esitab projektide projekti.

3.1.2 Töö projektifailidega

Olgu mingis tavalises tööaknas A avatud projektikirjeldus, st. projektikirjeldust esitav skeemtekst.

Olemasoleva projektifaili avamine: Projektifaili saab avada, kui vastav *Open-script* sisaldab vajalikku avamistegevust, nt *Import: ...* . Avada saab kas uude, või siis mingisse olemasolevasse projektifaili aknasse.

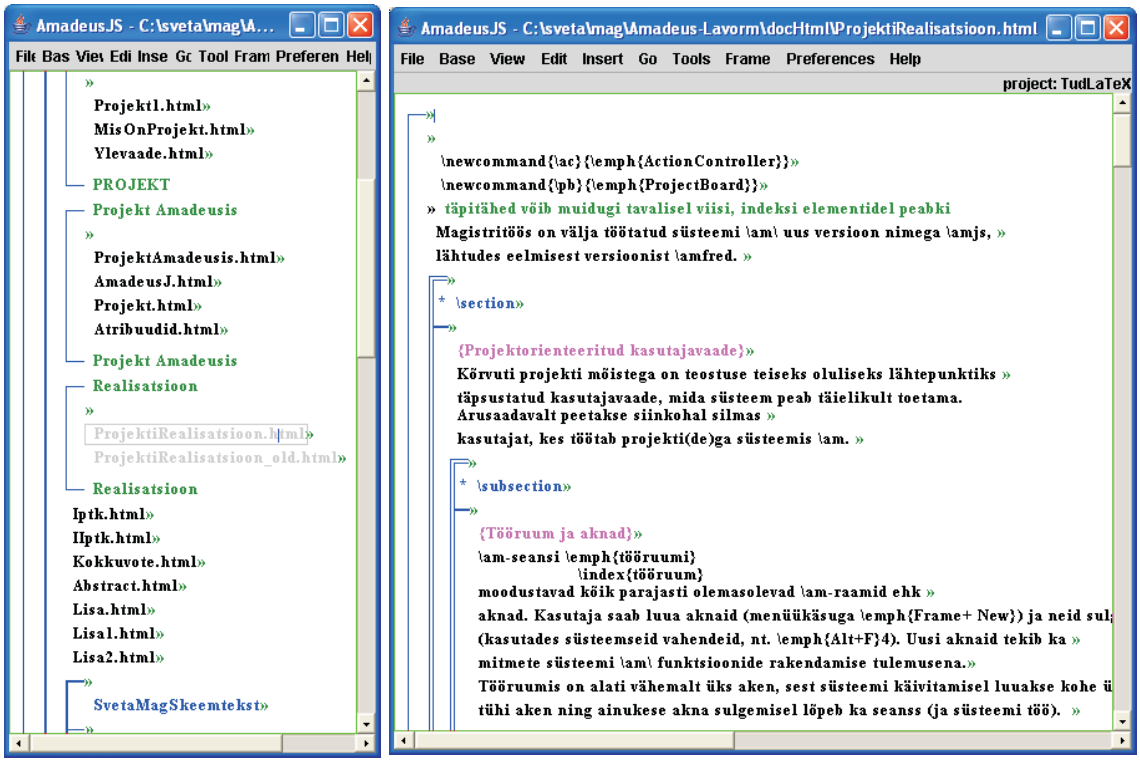
[I] Teha hiire kolmikklõps projektifaili nimel (nt. 'f.html') projektikirjelduses (aknas A): luuakse uus projektifaili aken ja täidetakse projekti (kuhu kuulub 'f.html') *Open-script*. Kui viimases on ette nähtud tegevus *Import: ...*, siis imporditaksegi fail 'f.html' loodud aknasse. Projektikirjelduse aknas värvitakse projektifaili nimi 'f.html' halliks. Projektikirjelduse aken ja projektifaili aken paigutatakse ekraanil kõrvuti, esimene väiksema laiusega (vt. joonis 3.2).

[II] Valida projektikirjelduses faili nimi (hiire üksik- või kaksikklõpsuga) ja seejärel teha mingis olemasolevas projektifaili aknas B menüükäsk *File+Open*. Sooritatakse samad tegevused, mis esimeselgi juhul, ainult uue akna asemel kasutatakse nüüd akent B.

Kui projektifail on juba avatud mingis aknas, siis hiire kolmikklõps selle faili (hallil) nimel projektikirjelduses lihtsalt toob selle projektifaili akna ekraanil esiplaanile.

Uue projektifaili loomist on selgitatud edasises, seoses projektikirjelduse haldamisega.

Projektifaili sulgemine: Sulgeda projektifaili aken süsteemset vahendit kasutades. Projektikirjelduse aknas taastub projektifaili nime värv (hallist mustaks).



Joonis 3.2: Projekti ja projektifaili aknad.

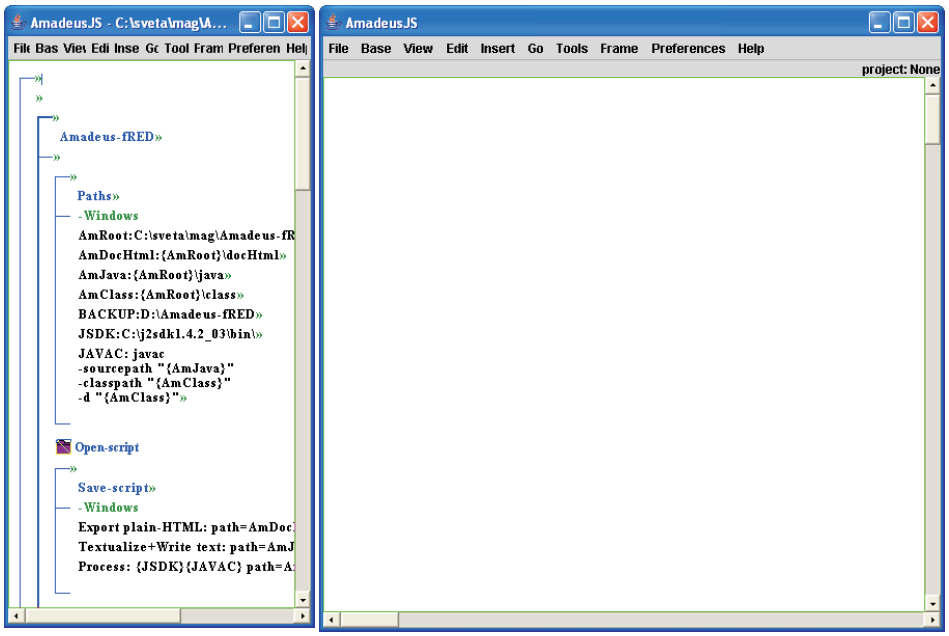
Projektifaili salvestamine: Projektifaili saab salvestada (1) projektikirjeldusest sõltumata või (2) menüükäsu *File+Save* vahendusel. Esimesel juhul rakendatakse *File*-menüüst sobivat valikut (*Write text*, *Export* ...) . Teisel juhul toimub salvestamine, kui vastav *Save-script* sisaldab ka vajalikku salvestustegevust, nt. *Export: ...* .

Projektifailiga seonduv muu töötlus on kirjeldatud atribuudirühmades *Open-script* ja *Save-script* ning käivitub, kui projektifaili aknas valitakse vastavalt kas menüükäsk *File+Open* või *File+Save*. Viimase asemel on võimalik on kasutada ka käske *File+SaveAll* ja *File+SaveAs*.

3.1.3 Projektikirjelduse haldamine

Projektikirjelduse avamine:

[I] Suvalises aknas rakendada menüükäsku *File+Import project*. Kasutajalt küsitakse projektikirjelduse faili nimi. Luuakse kaks akent: projektikirjelduse aken ja projektifaili aken. Esimesse sisestatakse skeemtekst projektikirjelduse failist, teine aken on tühi ja selles projektikirjelduse nimeks *None*. Projektikirjelduse aken ja projektifaili aken paigutatakse ekraanil kõrvuti, esimene väiksema laiusega (vt. joonis 3.3).



Joonis 3.3: Avatud projektikirjeldus ja tühi projektifaili aken.

[II] Suvalisse aknasse importida vastavast failist projektikirjelduse skeemtekst. Sel korral akende ümberpaigutamist esialgu veel ei toimu.

Projektikirjelduse muutmine ja salvestamine:
Kuna projektikirjeldus paikneb tavalises tööaknas, siis toimetamine ja salvestamine sooritatakse tavalisel viisil. Salvestamiseks sobib näiteks *File+Export plain-HTML*. Jooksvat projektikirjeldust võib korrigeerida; tehtud muudatused jõustuvad kohe, st. järgneval antud projektikirjeldusse kuuluvate projektifailide töötlusel kasutatakse juba korrigeeritud projektikirjeldust.

Uue projektikirjelduse loomine:

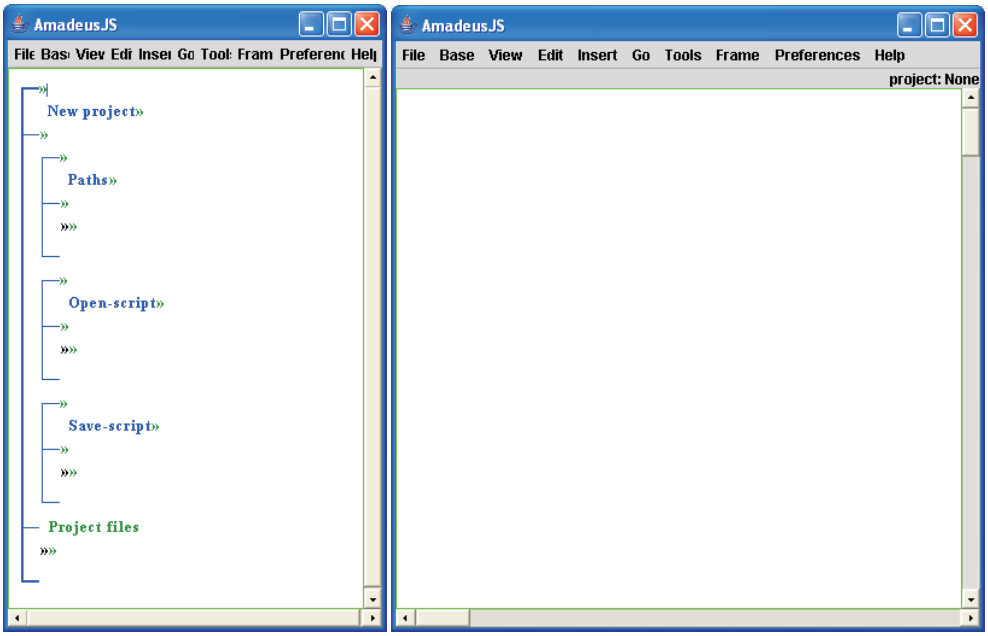
[I] Projektikirjelduse võib luua suvalises aknas, nö. nullist alates.

[II] Menüükäsu *File+Import project* rakendamisel anda ette uue projektikirjelduse veel mitte eksisteeriva faili nimi. Siis luuakse kaks akent: projektikirjelduse aken ja projektifaili aken. Esimesse sisestatakse projektikirjelduse mall, teine aken on tühi ja selles projektikirjelduse nimeks *None*. Projektikirjelduse aken ja projektifaili aken paigutatakse ekraanil kõrvuti, esimene väiksema laiusega (vt. joonis 3.4).

Uue projektifaili loomine: Lisada projektikirjeldusse uue faili nimi ja teha sellel hiire kolmikklõps. Pärast uue faili loomise soovi kinnitamist dialoogiaknas avatakse uus projektifaili aken tühja skeemtekstiga (mis ongi uue projektifaili esialgseks sisuks). Muidugi võib uue projektifaili luua mingis tavalises tööaknas ning salvestada. Seejärel lisada projektikirjeldusse uue faili nimi ning töötada edasi nagu olemasoleva projektifailigagi, nt. teha faili nimel hiire kolmikklõps faili avamiseks.

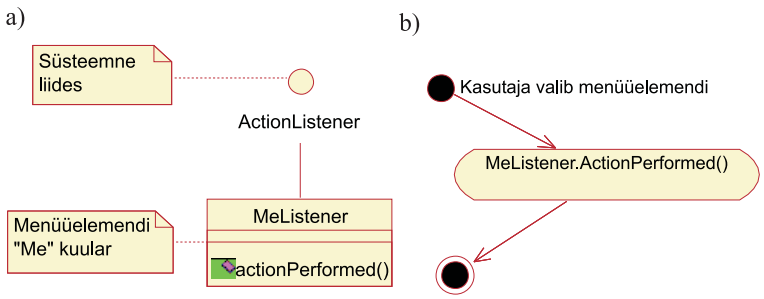
3.2 Lähteversiooni refaktoreerimine

Projektitööga seotud funktsionaalsuse teostuse huvides osutus vajalikuks lähteversiooni *Amadeus-fRED* struktuuri mõningane ümberkorraldamine. Refaktoreerimine puudutab eeskätt menüükäskude täitmise skeemi.



Joonis 3.4: Projektikirjelduse mall.

Iga menüükäsu ehk menüüelemendiga (*MenuItem*) on seotud vastava kuulariklassi isend, mille meetod *ActionPerformed* käivitub selle menüüelemendi valimisel kasutaja poolt. Versioonis *Amadeus-fRED* sisaldab see meetod *vahetult* kõik vastava menüükäsu täitmiseks vajalikud tegevused (vt. joonis 3.5), sh. muidugi ka vajalike abimeetodite rakendamise.



Joonis 3.5: Menüüelemendi *Me* kuular (a) ja töö skeem (b) lähteversioonis.

Uues, refaktoreeritud versioonis toimub mõnede menüükäskude täitmine

vahendatult. Vahendatult täidetavateks on muudetud kõik need funktsioonid, mille rakendamine on lubatud ka projektikirjeldusest, tegevuste rühmadest *Open-script* ja *Save-script*. Vahendatult toimub nt. menüükäskude *File+Export plain-HTML* ja *Tools+Textualize* täitmine. Taolisi käske saab aktiveerida nii menüüst (endistviisi) kui ka aknast *File+Open* ja/või *File+Save* käskude kaudu (muidugi juhul, kui *Open-* või *Save-script* rühmas rakendub mõni nendest käskudest).

Vahendustegevuse realiseerib uus klass *ActionController*[1], mille meetodisse *execute* on üle toodud kõigi vahendatult täidetavate menüüelementide kuularite meetodi *ActionPerformed* sisud. Iga sellise kuulari meetodisse *ActionPerformed* on kirjutatud nüüd ainult:

ActionController.execute(tegevuse_nimetus, frame),

kus *tegevuse_nimetus* on klassi *ActionController* konstant, mis vastab menüüelemendi nimetusele ja *frame* – projektiaken, milles tegevus välja kutsuti.

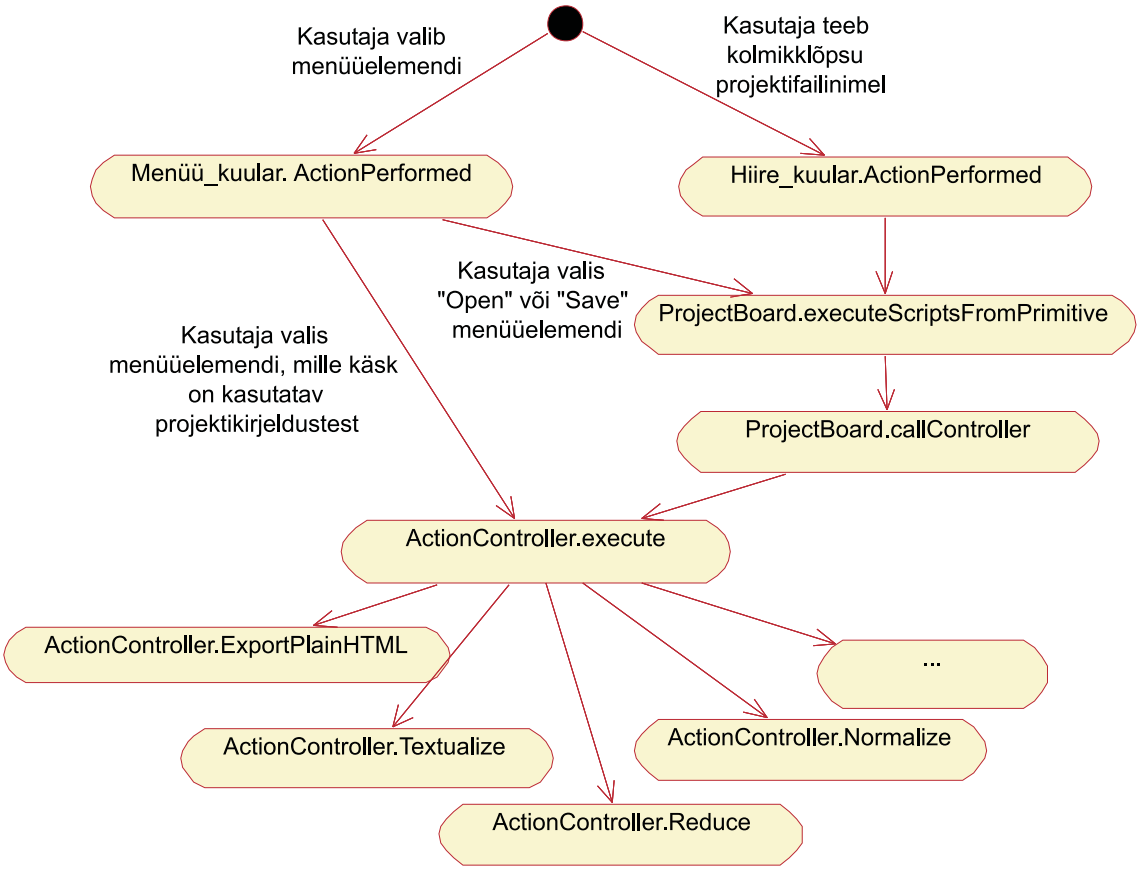
Uude versiooni on lisatud ka uus klass *ProjectBoard*[2], kus asuvad kõik süsteemi *Amadeus* projektitöö realisatsiooniga seotud meetodid. Klassi *ActionController* meetodeid kasutavad nii menüüdega seotud kuularid kui ka klassi *ProjectBoard* meetodid. Üldiselt, selleks et süsteemi mingi funktsioon oleks välja kutsutav ka projektikirjeldusest, peab seda funktsiooni realiseeriv meetod (täpsemalt: meetodi *actionPerformed* endine sisu) olema viidud klassi *ActionController*. Klasside *ProjectBoard* ning *ActionController* koostöö ja meetodite järjestikune väljakutsumine on näidatud joonisel 3.6.

3.3 Klass *ActionController*

Nagu eespool selgitatud, on vahendatud kuularite funktsionaalsust täitvad lähteversiooni meetodid kopeeritud klassi *ActionController* vastavateks meetoditeks. Et menüükäsud töötaksid korrektselt ka projektikirjeldustest rakendamisel, on neid täiendatud järgmiselt.

- Meetodi alguses tuvastatakse, kas meetod kutsuti välja projektikirjeldusest või mitte. Selleks on klassis *ActionController* kontrollimismeetod

[1]Klassi *ActionController* täpsem kirjeldus on esitatud jaotises 3.3.
[2]Klassi *ProjectBoard* täpsem kirjeldus on esitatud jaotises 3.4.



Joonis 3.6: Süsteemi kasutajaliidese ja klasside *ProjectBoard*, *ActionController* koostöö.

isCalledFromProjectBoard, mis tagastab *true*, kui funktsioon on välja kutsutud klassist *ProjectBoard*.

- Kui meetod ei ole rakendatud klassist *ProjectBoard*, siis meetodis eritegevusi ei tehta. Kui aga meetod on rakendatud klassist *ProjectBoard*, siis tuleb eraldi vaadelda järgmisi situatsioone.

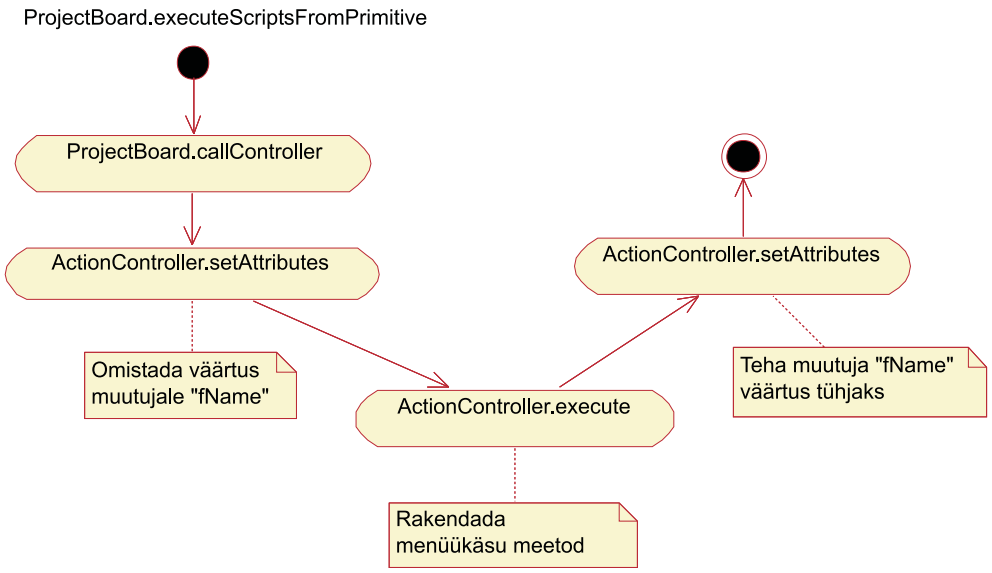
  [**1**] Menüükäsk töötab faili nimega (loeb failist, impordib failist, ekspordib faili, ...) – menüükäsk peab kasutama faili nime (mis on salvestatud klassi *ActionController* muutujasse *fName*).

  [**2**] Menüükäsu tulemus kuvatakse uues aknas (nt. jooksev skeemtekst

tekstualiseeritakse) – meetodi töö peab toimuma samas projektifaili raamis (raam on antud meetodile parameetrina).

[3] Menüükäsu täitmisel joonistatakse raami pilt ümber – meetodis ei ole vaja raami pilti ümber joonistada; klass *ProjectBoard* hoolitseb selle eest ise.

Klass *ActionController* sisaldab staatilist muutujat *fName*, mille väärtus on muudetav ainult klassist *ProjectBoard*. Muutuja otstarbeks on säilitada omistatud väärtust (tavaliselt faili nimi) klassist *ProjectBoard* rakendatud *ActionController* meetodi *execute* töö ajal. Joonisel 3.7 on näidatud, millal klass *ProjectBoard* muudab muutujat *fName*.



Joonis 3.7: Meetodi *ProjectBoard.callController* töö.

Klass *ActionController* sisaldab samuti ka konstante, mille väärtused vastavad süsteemi *Amadeus* menüüelementide alatõstus (inglisekeelsetele) nimedele. Näiteks menüüelementidele *Export HTML*, *Read text*, *Textualize* vastavad järgmised konstandid:

```
public static final String EXPORT_HTML = "export html";
public static final String READ_TEXT = "read text";
public static final String TEXTUALIZE = "textualize";
```

Klassi *ActionController* põhimeetod

void execute(String action, AmFrame frame)

rakendab sõltuvalt esimese parameetri väärtusest (võrdleb seda klassi *ActionController* konstantidega) vajalikku funktsiooni realiseeriva meetodi. Menüüelementide kuularid kasutavad samuti neid konstante, nimelt parameetritena klassi *ActionController* meetodi *execute* väljakutsumisel. Klassist *ProjectBoard* kutsutakse klassi *ActionController* meetod *execute* välja tegevus-atribuutides antud menüükäskude abil.

3.4 Klass *ProjectBoard*

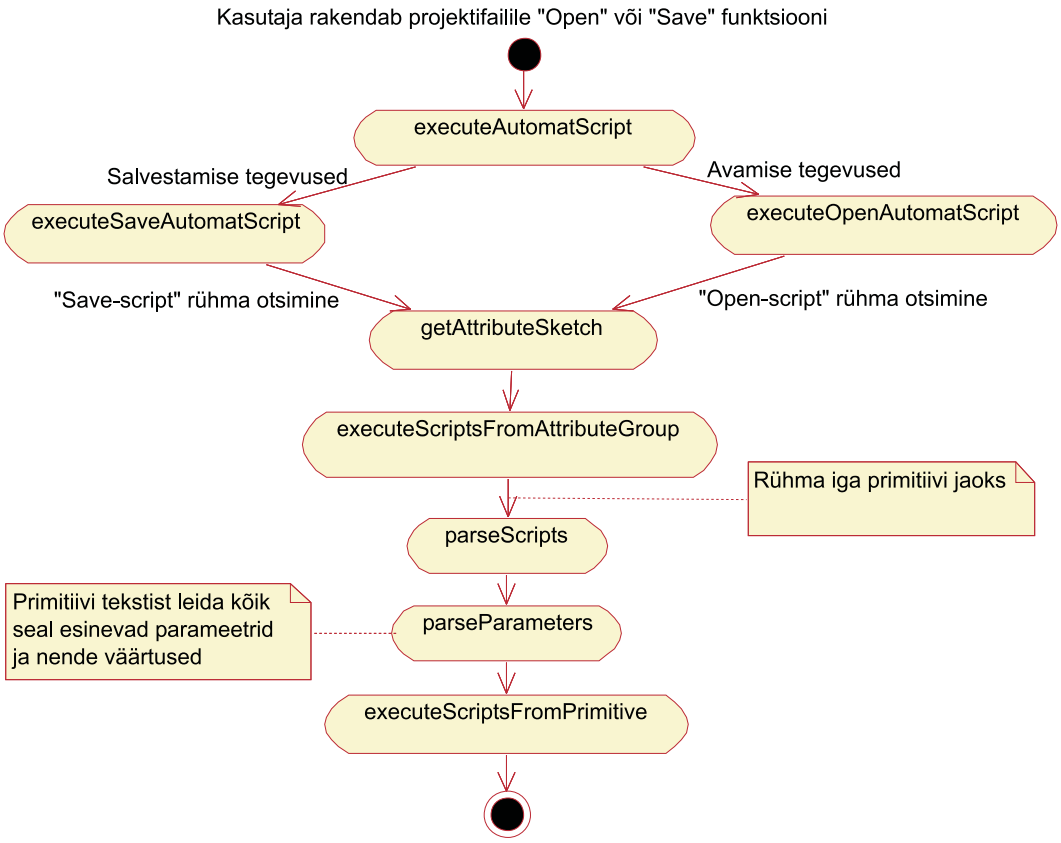
Klassi *ProjectBoard* põhiülesandeks on süsteemi ja projektikirjelduse koostöö organiseerimine. Lühidalt öeldes – käivitada mingi projektikirjelduse tegevuste rühmades paiknevaid menüükäske. Klassi *ProjectBoard* põhimeetodite rakendamise järjestus on näidatud joonisel 3.8.

Käesolevas töös teostatud versioon *AmadeusJS* toetab kahte tegevuste rühma: *Open-script* ja *Save-script*. Nagu näha jooniselt 3.8, esineb ühe tegevuste rühma töötlemine klassis *ProjectBoard* ainult ühes kohas. Meetodites *executeOpenAutomatScript* ja *executeSaveAutomatScript* otsitakse projektikirjelduse vastavad tegevuste rühmade skeemid (*Open-script* ja *Save-script* rühmad), samuti toimuvad eri tegevused avamise ja salvestamise jaoks. Järelikult saab hõlpsasti teostada ka uut liiki tegevuste rühmi, näiteks kui soovitakse laiendada projektikirjelduse mõistet mõnes järgmises süsteemi *Amadeus* versioonis.

Kui tegevuste rühmade skeemid on leitud ja eritegevused tehtud, siis edasine töö toimub jälle ühe skeemi järgi. Meetod *executeScriptsFromAttributeGroup* analüüsib tegevuste rühma skeemi iga primitiivi: leiab funktsioonide nimed (*parseScripts*) ja parameetrite väärtused (*parseParameters*). Seejärel meetodist *executeScriptsFromAttributeGroup* kutsutakse välja meetod *executeScriptsFromPrimitive* iga primitiivi jaoks.

Meetodi *executeScriptsFromPrimitive* järjestikuste tegevuste skeemi selgitab joonis 3.9.

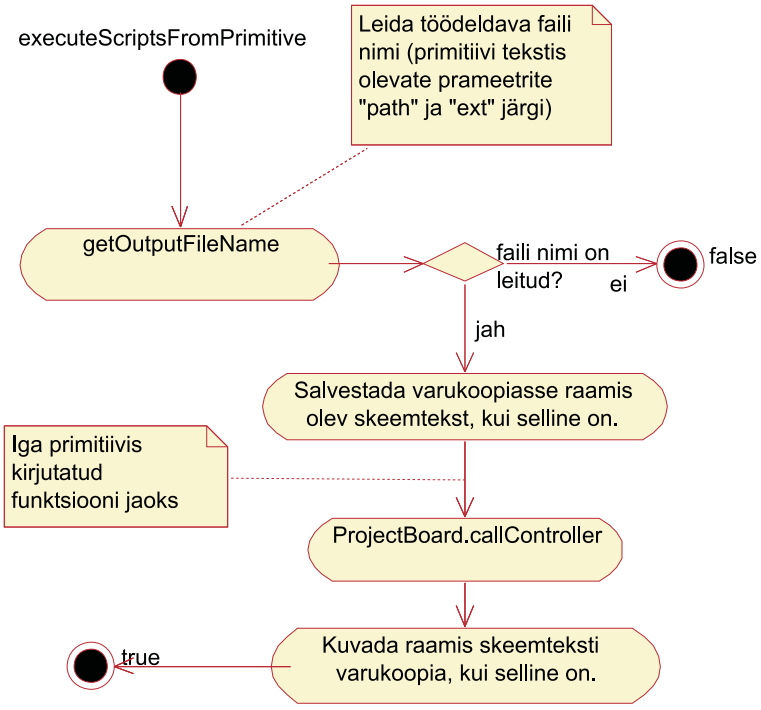
Tegevus-atribuudi defineerimisel saavad parameetrite väärtusteks olla



Joonis 3.8: Klassi *ProjectBoard* põhimeetodite skeem.

Path rühma atribuutide nimed. Parameetrite tegelike väärtuste otsimine toimub joonisel 3.10 asuva skeemi järgi. Kui antud nimega atribuuti ei leita nii jooksva kui ka ülemiste tasemete projektikirjelduste *Path* rühmades, siis tegevus-atribuudi parameetri väärtus jääb asendamata.

Meetodi *getAttrElementInherited* detailsem kirjeldus paikneb joonisel 3.11. Kõigepealt toimub *Path* rühma skeemi otsimine projektikirjelduses. Meetod *getAttributeSketch* otsib atribuudirühma, alustades antud projektikirjeldusest, ja seejärel kõikidest ülemprojektikirjeldustest, kuni skeem on leitud või kuni kõik võimalikud projektikirjeldused on läbi vaadatud. Seega, kui meetod *getAttributeSketch* ei leia atribuudirühma skeemi, siis järelikult atribuuti ei leidu ja meetod *getAttrElementInherited* tagastab sel juhul väärtuse *null*.

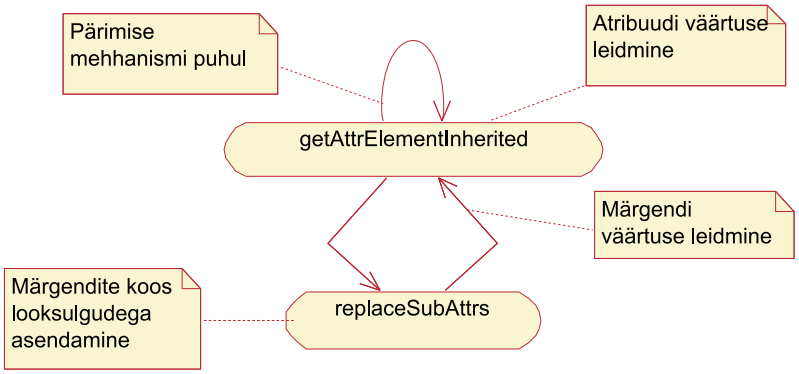


Joonis 3.9: Klassi *ProjectBoard* meetodi *excuteScriptsFromPrimitive* skeem.

Kui tegevus-atribuut defineerib *Process* käsu, siis parameetrite analüüs toimub mitte meetodis *parseParameters*, vaid meetodis *parseProcessParameters*. Meetod *parseProcessParameters*, vastavalt selgitusele jaotises 2.2.3, asendab parameetrite tekstis kõik looksulgudes leiduvad märgendid vastavate atribuutide väärtustega, ja parameetrite *path* ja/või *ext* esinemise kohale paigutab moodustatud faili nime.

3.5 Automaatne salvestamine

Kuna süsteem *Amadeus* ei ole veel piisavalt stabiilne, siis on väga oluline automaatse salvestamise olemasolu – jooksvalt toimetatavate failide perioodiline salvestamine ajutisse kataloogi. Sellega tagatakse värskete varukoopiate kättesaadavus pärast süsteemi töö ootamatut lõppu (nii tarkvaralistel kui ka riistvaralistel põhjustel). Kõrvuti projektitöö võimalustega suurendab auto-



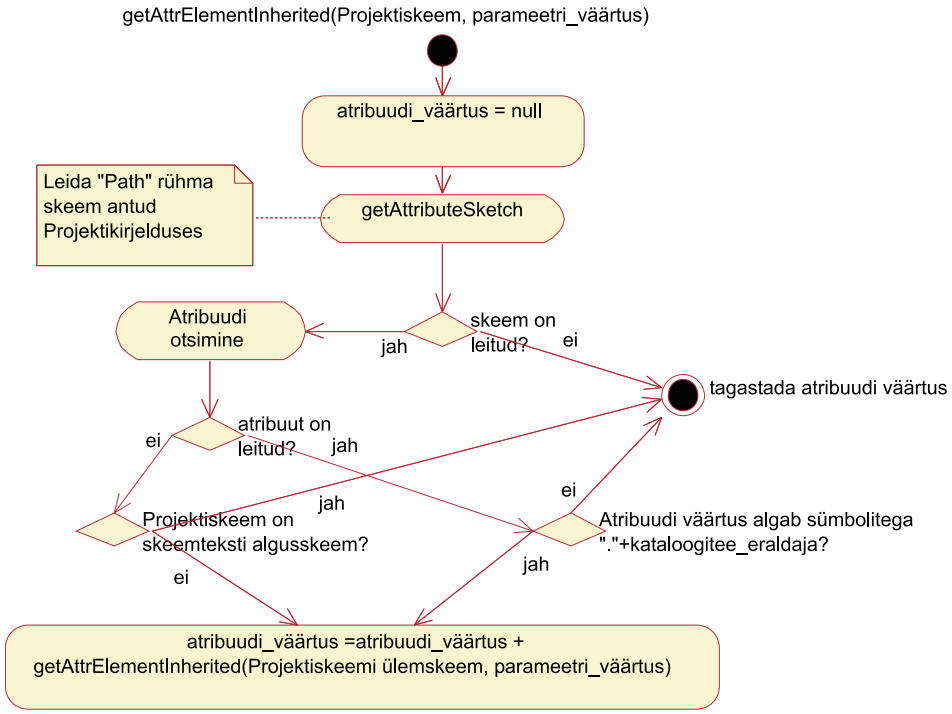
Joonis 3.10: Atribuutide väärtuste leidmine.

maatsalvestus oluliselt kogu süsteemi kasutuslikku efektiivsust.

Automaatne salvestamine rakendub skeemtekstile, millega on seotud faili nimi (ei pea olema projektifaili nimi). Skeemteksti muudatused salvestatakse vaikimisi iga minuti järel. Kuid salvestamise soovitud tiheduse saab seada süsteemi *Amadeus* omaduste failis "Amadeus.properties" omaduse *autoSaveTimePeriod* abil.

Automaatne salvestamine toimub menüüfunktsiooni *export-XML* abil, kuna see annab võimaluse salvestada skeemteksti koos kõigi detailidega. Skeemtekst auto-salvestatakse süsteemi *Amadeus* omaduste failis omadusega *tempDirectory* määratud *temp* kausta. Kui kasutaja rakendab skeemtekstile mingit eksportfunktsiooni või üldfunktsiooni *Save*, siis vastav fail kaustas *temp* kustutakse.

Faili nimele mingi avamise funktsiooni rakendamisel annab süsteem võimaluse avada selle asemel hilisem (värskem) auto-salvestatud selle faili versioon (kui selline leidub). Kui kasutaja loobub sellest võimalusest, siis vastav fail kustutakse kaustast *temp*.



Joonis 3.11: Meetodi *getAttrElementInherited* tegevuste järjekord.

Kokkuvõte

Praktiliselt kõik integreeritud arenduskeskkonnad ja isegi mõned lihtsamad tekstitoimetid toetavad abivahendit, mida nimetatakse *projektiks*. Projekti olemasolu mingis arenduskeskkonnas lihtsustab kasutaja tööd, kiirendades eeskätt just neid protsesse, mis ei ole vahetult toimetamisega seotud.

Süsteemi *Amadeus* edasiarendamise käigus ilmnes samuti vajadus projekti järele. Kasutaja kordab ühetaolisi töötlusfunktsioone (või isegi funktsioonide jadasid) iga töödeldava faili jaoks enne ja pärast skeemteksti toimetamist. Multifunktsionaalses süsteemis on aga funktsioonide valik üsna lai ja kasutaja peab tihti otsustama (ja meeles pidama), milliseid funktsioone ta rakendab konkreetset tüüpi faili töötlemiseks. Mingil perioodil regulaarselt kasutatavate failide jaoks oleks väga soodne automatiseerida töötlusfunktsioonide rakendamine.

Käesolevas töös on antud ülevaade projekti mõistest ja selle realiseerimisest mõnedes arenduskeskkondades. Ülevaatest nähtub, et erinevates arenduskeskkondades mõistetakse projekti all üsnagi erinevaid objektide ja atribuutide komplekte. See on arusaadav, kuna projekt on realiseeritud vastavalt oma spetsiifilise keskkonna nõudmistele ja vajadustele. Samuti kasutatakse projektikirjelduste esitamiseks erinevaid keeli (süntaksit) .

Antud töö põhitulemuseks on mõiste *projekt* täpne defineerimine ja sellele vastav teostamine mitmevaatelises mitmekeelses üldfunktsionaalses süsteemis *Amadeus*.

Süsteemi *Amadeus* jaoks on välja töötatud sobiv projekti definitsioon ja detailne kirjeldus. Käesoleva töö aluseks olevas süsteemi *Amadeus* lähteversioonis olid realiseerimata niisugused harilikud funktsioonid nagu *Avada* ja *Salvestada*. Näiteks tuli kasutajal toimetatava skeemteksti salvestamiseks

iga kord valida mitme võimaliku salvestusformaadi vahel. Projekt süsteemis *Amadeus* lahendab nii mõnegi probleemi, lubades kasutajal endal valida, missugused failinimed kuuluvad projektisse ja määrata, missuguseid süsteemis olevaid töötlusfunktsioone rakendada faili avamisel ning salvestamisel.

Projekti teostusega realiseeriti antud töös just avamise ja salvestamise funktsioonid. Vastava menüükäsu (*Open* või *Save*) täitmisel aga ei teostata lihtsalt (teistes süsteemides tavapärased) faili avamise või salvestamise operatsioonid, vaid mõlemal juhul vallandub terve operatsioonide jada. Viimase kirjeldus aga võetakse vastavast projektikirjeldusest. Seega kasutaja, koostades projektikirjelduse, määrab ise kindlaks, millised tegevused tuleb sooritada antud projekti kuuluvate failide korral, kui neile rakendatakse käsku *Open* või *Save*. Loomulikult on käsu *Open* tegevuste jadas (atribuudirühmas *Open-script*) üheks tegevuseks faili importimine ja käsu *Save* tegevuste jadas (atribuudirühmas *Save-script*) üheks tegevuseks faili eksportimine kindlas formaadis.

Projektikirjeldus koostatakse skeemteksti vormis, sellesse saab lülitada ka vastava arendusprojekti enda kirjelduse või mõne osa sellest. Või ka vastupidi – arendustöö projektikirjeldus esitatakse skeemtekstina S, kuhu on lülitatud projektikirjeldused süsteemi *Amadeus* mõttes. Ka viimasel juhul on skeemtekst S muudatusteta kasutatav kui süsteemi *Amadeus* projektikirjeldus. Lisaks muudele avamise ja salvestamise tegevustele saab arendustöö projekti korral nt. üheks avamistegevustest määrata sissekande tegemise projekti logiraamatusse (aeg, millal asuti selle failiga tegelema) ja üheks salvestustegevustest – samuti sissekande tegemine logiraamatusse (aeg, millal selle faili uus versioon valmistati ja läks järjekordsele testimisele).

Amadeus-projekti iseloomustab hierarhilisus. Töös on välja töötanud ka vastav pärilusmehhanism – moodus, kuidas alamprojektid pärivad atribuute ülemprojektidest.

Projekti realiseerimiseks on valitud skeemtekstide keel, kuna skeemteksti hierarhiline struktuur sobib selleks väga hästi. Projektikirjeldusi käideldakse tavalises *Amadeus*-aknas ja selle toimetamine ei erine suvalise skeemteksti toimetamisest. Projektikirjelduse muudatused jõustuvad koheselt. Järgneval antud projektikirjeldusse kuuluvate projektifailide töötlusel kasutatakse juba korrigeeritud projektikirjeldust. Projektikirjelduse koostamise reeglid on töös

esitatud vastava skeemmudelina.

Töö otsene tarkvaraline tulemus on süsteemi *Amadeus* uus versioon *AmadeusJS*. Viimases on lisaks projekti teostusele realiseeritud ka skeemteksti automaatse salvestamise võimalus.

Antud töö nii programmeerimise osa kui ka magistritöö tekst on kirjutatud süsteemi *Amadeus* abil ja vastavate projektide toel.

Project and its implementation in the system *Amadeus*

Master thesis

Svetlana Golovko

Abstract

Any common homogeneous IDE supports (file) projects. Project in IDE is a tool for managing logically bounded files. In the case of heterogeneous and multifunctional IDEs, in which files' action attributes may essentially vary from file to file, the role of project tools becomes extremely important.

In this thesis, the problem of definition and implementation of file projects is considered. In particular, the problem for the target system *Amadeus* is solved. *Amadeus* is a software tool for handling graphically modeled computer texts, so called sketchy texts.

First of all, an overview concerning projects in several common IDEs (Eclipse platform, Microsoft Visual C++, JCreator, ...) is given and some differences and similarities between them are pointed out. On the basis of the analysis, a precise definition of a file project for a multifunctional environment (like *Amadeus*) is given. In a project, every file name is assigned an attribute set (set of actions or variables, that is assigned to a file name by the file project). One attribute set can be assigned also to a group of file names. Variable attributes usually define paths for a project file. Action attributes define system function (or list of functions) which relate to a group of project

files.[3] To provide attribute inheritance mechanism the notation of subproject in a file project is introduced.

The main result of this work is file project implementation in the multi-functional development environment *Amadeus*. Earlier, there was no possibility to use common Open and Save functions in *Amadeus*. Any of environment functions had to be separately chosen and invoked for a file. Provided in this work the project implementation for *Amadeus* resolves this problem. User can add file names to a project description and also describe which system functions should be called in connection with Open and Save operations applied for the files from the project description.

In *Amadeus*, there is no need to introduce a special language for project descriptions. Any project description is represented as a project description sketch in a usual *Amadeus* window. Indeed, while creating project description sketches certain representation rules should be followed. For example, any project description sketch is represented by a module sketch with just two simple branches, one for an attribute subset and another for a set of file names and/or subprojects.

New version named *AmadeusJS* of the system *Amadeus* is released. File and Tools menu functions from the previous version (*Amadeus-fRED*) are made available from project descriptions in *AmadeusJS*.

[3]Environment functions here are all export and import functions (Read Text, Write text, Print plain-Text, Print PostScript) from File menu and all functions from *Amadeus* Tools menu.

Kirjandus

[1] J.Kiho, S.Solopova. *Heterogeneous File Project 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

[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. Pp. 151-162.

[3] Jüri Kiho. *Sketchy Modeling of Computer Texts*. Research report, 18th January 2000, 64p.

[4] EditPlus TextEditor v2.11 abi-info dokumentatioon

[5] MSDN Library Visual Studio 6.0

[6] Nick Edgar, Kevin Haaland, Jin Li, and Kimberley Peter *Eclipse User Interface Guidelines, Version 2.1*, February 2004, http://www.eclipse.org/articles/Article-UI-Guidelines/Contents.html

[7] JCreator 3.0 abi-info dokumentatioon

[8] Borland Delphi 5.0 abi-info dokumentatioon

[9] WinShell 2.5 abi-info dokumentatioon

Indeks

Lisa
Loodud tarkvara (CD-l)

[I] Süsteemi *Amadeus* versioon *AmadeusJS* (kaust *AmadeusJS*).

[II] Käesoleva töö tekst (*MagToo.pdf*).

[III] Süsteemi *Amadeus* arendamisel ning käesoleva magistritöö teksti koostamisel kasutatud projektikirjeldused (kaust *Portfolio*).