VERSTAS-PROJEKTI

Sovellusraportti

Antti Hakala

Tomi Laamanen

Paavo Nieminen

Jukka Toivanen

6.1.2003

Jyväskylän yliopisto

Tietotekniikan laitos

Tekijät Antti Hakala, Tomi Laamanen, Paavo Nieminen, Jukka Toivanen

Yhteystiedot verstas@korppi.jyu.fi, anthakal@cc.jyu.fi, laamanen@cc.jyu.fi, nieminen@cc.jyu.fi, jitoivan@cc.jyu.fi

Projektitila Agora C223.4

Puhelin 014-2604966

Kotisivu http://kotka.it.jyu.fi/verstas/

Työn nimi Verstas-projektin sovellusraportti

Työ Tietotekniikan sovellusprojekti

Tiivistelmä:

Tämä dokumentti on Jyväskylän yliopistossa syksyllä 2002 toteutetun Verstas-sovellusprojektin sovellusraportti. Raportissa kuvataan sovelluksen rakennetta yleisesti, esitellään ohjelmaosioiden toiminta ja rajapinnat sekä ohjelman sisäiset tietorakenteet. Lisäksi esitellään sovelluksen käyttöliittymää.

Avainsanat: mallinnusympäristö, käyttöliittymä, sovelluskäsikirja

Versiohistoria

Versio	Pvm	Tekijä	Kuvaus
0.1	30.11.2002	JT	Ensimmäinen raakaversio. Termiluettelo, asiaa vaatimuksista ja koodin kommentoinnista.
0.2	12.12.2002	JT	Lisätty luokkien ja metodien kuvauksia Sovelluksen luokat käännösyksiköittäin -kohtaan.
0.3	15.12.2002	TL, PN	Lisätty käyttöliittymäkuvat ja projektimanageriin liittyvät käännösyksiköt
0.4	16.12.2002	PN	Lisätty projektimanagerin luokkakaaviot
0.5	17.12.2002	JT	Lisätty loppujen käännösyksiköiden luokat ja korjattu virheitä.
0.6	18.12.2002	JT	Lisätty sekvenssikaavioita ja suunnittelumallien kuvauksia.

Tekijöiden lyhenteet

AH Antti Hakala

TL Tomi Laamanen

PN Paavo Nieminen

JT Jukka Toivanen

Hyväksytty

Pvm .... Allekirjoitus ....

Pvm .... Allekirjoitus ....

Sisältö

1 Johdanto

Verstas-projekti suunnitteli ja toteutti Numerola Oy:lle yhtenäisen ja kokoavan käyttöliittymän mallinnus- ja simulointiohjelmistoille. Projekti toteutettiin Jyväskylän yliopiston tietotekniikan sovellusprojektina. Projektiryhmään kuuluivat Antti Hakala, Tomi Laamanen, Paavo Nieminen ja Jukka Toivanen. Projektin vastaavana ohjaajana toimi Kari Kärkkäinen ja teknisenä ohjaajana Ville Tirronen.

Tässä raportissa kuvataan projektin puitteissa toteutetut sovellukset Verstas ja Verstas Solver. Luvussa 2 esitellään sovelluksille asetettuja vaatimuksia ja niiden toteutumista. Luku 3 esittelee sovelluksia ohjelmoitaessa käytettyä kommentointi- ja nimeämiskäytäntöä. Luvussa 4 kuvataan Verstaan ja alisovellusten välistä kommunikointia varten määritellyt rajapinnat ja luvussa 5 Verstaan ja Verstas Solverin käyttämien tiedostojen rakenne. Luvussa 6 esitellään Verstaan arkkitehtuuria ja luvuissa 7 sekä 8 Verstaan ja Verstas Solverin luokat käännösyksiköittäin. Lopuksi luvussa 9 esitellään sovelluksen käyttöliittymää ja luvussa 10 listataan ideoita jatkokehittäjälle.

1.1 Termit ja käsitteet

1.1.1 Sovellukset, kirjastot ja tiedostot

Alisovellus: Numerolan toteuttama sovellusohjelma, jolla toteutetaan yksi mallinnusprosessin vaiheista.

Dmg: Alisovellus, joka muuttaa 2D-geometrian laskentaverkoksi.

Emacs: Laajennettava ja muokattava tekstieditori.

Exodus: Alisovellus, työkalu 3-ulotteisen laskentageometrian luomiseen.

Genesis: Alisovellus, työkalu 2-ulotteisen laskentageometrian luomiseen.

Makefile: Make-sovelluksen käyttämä tiedosto ohjelman lähdekoodin kääntämisen hallintaan.

Mbgen: Alisovellus, joka muuttaa 3D-geometrian laskentaverkoksi.

Numerrin-ydin: Numerolan toteuttama FEM-aliohjelmakirjasto.

Plot: Alisovellus visualisointiin.

TeeKL, AutoUI: Alisovellus, työkalu graafisen käyttöliittymän automaattiseen tuottamiseen ja hallintaan.

Verkkogeneraattori: Alisovellus, joka tuottaa geometriasta laskentaverkon. Verstaan alisovellusten käytössä on useita eri tilanteisiin sopivia verkkogeneraattoreita, kuten Verkotin, Dmg, Mbgen.

1.1.2 Ohjelmointikielet ja -ympäristöt

Elisp: Emacs lisp, Emacsin laajentamiseen käytettävä ohjelmointikieli.

Fortran: Erityisesti laskennallisten ongelmien ratkaisuun käytetty ohjelmointikieli.

Kylix: C++/Object Pascal -kehitysympäristö Linux-käyttöjärjestelmälle.

Object Pascal: Olio-ohjelmointilaajennos Pascal-ohjelmointikieleen.

1.1.3 Mallinnusprosessi ja laskenta

FEM: Finite Element Method, äärellisten elementtien menetelmä, eräs numeerinen menetelmä osittaisdifferentiaaliyhtälöiden numeeriseksi ratkaisemiseksi.

Geometrian luonti: Mallinnusprosessin vaihe, jossa tutkittavan fysikaalisen järjestelmän ominaisuudet pyritään kuvaamaan tietokoneohjelman avulla. Tuloksena on ns. laskentageometria.

Laskenta: Mallinnusprosessin vaihe, jossa käännetty laskentaohjelma ajetaan syötteenään laskentaverkko sekä parametrit, tulosteena saadaan laskentatulokset.

(Laskenta)geometria: Tietokoneella luotu tutkittavan fysikaalisen kappaleen muodon ja rakenteen kuvaus, esitettynä joukosta yksinkertaisia käyriä tai pintoja muodostuvana approksimaationa.

(Laskenta)geometrian luonti: Mallinnusprosessin vaihe, jossa kuvataan tutkittavan fysikaalisen järjestelmän geometriset ominaisuudet. Tuloksena on laskentageometria.

(Laskenta)malli: Matemaattinen malli, jonka ilmenemismuoto on fortran-lähdekoodi.

Laskentaohjelma: Fortran-kääntäjällä laskentamallista käännetty ajettavissa oleva ohjelma.

Laskentatulos: Laskentaohjelman tuloste, matemaattisen mallin ratkaisu annetuilla lähtötiedoilla (geometria, parametrit), käytännössä ASCII-tekstinä esitettyä numeerista dataa.

Laskentaverkko: Verkkogeneraattorin tuloste, geometrian jako laskentaa varten osiin esim. kolmioihin tai tetraedreihin.

Laskentaverkon luonti: Mallinnusprosessin vaihe, jossa laskentageometriasta tuotetaan verkkogeneraattorilla laskentaohjelman tarvitsemaan laskentaverkko.

Mallin toteutuksen etsintä: Valittua matemaattista mallia vastaavan ohjelman etsiminen tai luominen.

Mallin valinta: Mallinnusprosessin vaihe, jossa päätetään (yleensä kirjallisuuteen tai tutkimukseen nojaten) myöhemmissä vaiheissa käytettävä matemaattinen malli.

Mallinnusprosessi: Nykyaikaisen tuotekehityksen ja tutkimuksen osa-alue, jossa prototyypin rakentamisen ja kokeellisen mittaamisen sijasta tehdään kokeiluja matemaattisella mallilla, joka kuvaa järjestelmän toimintaa annetuissa olosuhteissa.

Optimointi: Simuloinnin (automaattinen) toistaminen jollakin kriteerillä geometriaa ja/tai parametreja muuttamalla, tavoitteena parempi lopputulos.

Parametrit: Laskentaohjelman syöte, esim. tutkittavan ilmiön fysikaaliset ominaisuudet.

Visualisointi: Laskentatulosten esitys ihmisen ymmärtämässä muodossa (käytännössä graafisesti erilaisin tavoin), eräs mallinnusprosessin vaihe.

1.1.4 Verstas

Mallinnusympäristö: Kokonaisuus, jossa kaikki mallinnuksen vaiheet voidaan tehdä yhtenäisen käyttöliittymän avulla hallittavilla työkaluilla.

Numerrin-Verstas: Verstas, Verstas-ympäristö. Mallinnusympäristö, jossa kaikki mallinnusprosessin vaiheet voidaan toteuttaa käyttäen Numerolan työkaluja ja Numerrin-ydintä.

Projekti: Kokonaisuus, joka sisältää kaikki tiettyyn simulointitehtävään liittyvät tiedot (s.o. mallit, geometriat, parametrit, tulokset, raportit jne).

Raportti: Tutkimus-, päivitys- tai muu raportti, esim. pdf-muodossa.

Tietokanta: Tässä projektissa hakemistorakenne sisältäen uudelleen hyödynnettäviä mallinnukseen käytettäviä tiedostoja.

1.2 Numerrin-järjestelmä

Numerrin-simulointiohjelmisto koostuu useista erillisistä sovelluksista (myöhemmin alisovelluksista). Verstas kommunikoi erillisten alisovellusten kanssa ja huolehtii niiden yhteiskäytöstä sekä syöttö- ja tulostustiedostojen hallinnasta. Myöhemmin tullaan tekemään uusia alisovelluksia liitettäväksi Verstaaseen, joten kommunikoinnin suunnittelussa on otettu huomioon myöhempi laajennettavuus.

Alisovellukset ottavat vastaan syötetiedostoja ja tulostavat tiedostoja, jotka jälleen menevät syötteeksi uusille alisovelluksille. Tärkeä osa Verstaan toiminnallisuutta on tiedostojen hallitseminen ja liittäminen asiaankuuluviin alisovelluksiin.

2 Vaatimukset ja niiden toteutuminen

Tässä luvussa kuvataan, kuinka vaatimusmäärittelyssä [6] ja käyttötapauksissa [7] esitetyt vaatimukset toteutuivat projektin puitteissa.

2.1 Sovelluksen päämäärä

Sovelluksen yleisenä päämääränä oli koota Numerolan Numerrin-simulointiympäristön eri ohjelmat yhdeksi mallinnusympäristöksi. Tämä toteutettiin kehittämällä yleinen sovellustenhallintajärjestelmä, johon voidaan vapaasti liittää mitä tahansa määritetyn TCP/IP -rajapinnan (kts. luku 4) toteuttava sovellus. Verstas ei siis ota lainkaan kantaa siihen, millaisia sovelluksia tai tiedostotyyppejä sen avulla käytetään, ja on näin ollen laajennettavissa lähes rajattomasti.

Verstas itsessään ei myöskään ole tietoinen itse mallinnusprosessista, vaan prosessin vaiheet ja rakenne määritellään Verstaan asetustiedostoissa. Kukin käyttäjä voi siis halutessaan muokata mallinnusprosessia omiin tarpeisiinsa sopivaksi muuttamalla pelkkiä asetustiedostoja. Vaatimusmäärittelyssä mainittuja käyttäjäryhmiä ei siis tarvinnut sisällyttää Verstaaseen, koska kukin käyttäjä voi muokata Verstaan toiminnallisuutta haluamakseen.

2.2 Käyttöliittymä

Käyttöliittymän yleisiksi vaatimuksiksi asetettiin käytettävyys ja helppo hallittavuus. Sovellus sisältää normaalitilassa ainoastaan yhden ikkunan, jonka avulla käyttäjä voi kontrolloida koko projektia. Lisäksi tilanteesta riippuen voi avoinna olla hakuun liittyviä dialogeja, asetusdialogi ja erinäinen määrä alisovelluksia.

Projektin sisältö esitetään puunäkymässä, jossa projektiin kuuluvat komponentit on jaoteltu hierarkisesti projektin vaiheiden mukaisesti. Käyttäjä voi näin nopeasti löytää ja aktivoida haluamansa projektiin liittyvän komponentin.

Käyttäjä voi asettaa tiedostotyyppien perusteella oletusarvoja siitä, kuinka komponenttia käsitellään Verstaassa. Komponentin voi mm. määrittää avautumaan automaattisesti tietyssä sovelluksessa heti kun komponentti liitetään projektiin. Myös muut komponentin asetukset määräytyvät käyttäjän oletusarvoihin tiedostotyypin mukaan, mutta niitä voi myöhemmin säätää komponenttikohtaisesti. Tällaisia asetuksia ovat mm. sovellus, jossa komponentti avataan, sekä komponentin tietojen automaattinen päivitys kaikkiin sitä käsitteleviin sovelluksiin kun komponentti talletetaan jossakin sovelluksessa.

Ikkunoiden asemoinnin helpottamiseksi toteutettiin Tile-, Cascade-, Minimize all- ja Restore-komennot, joilla ikkunat voidaan nopeasti asemoida uudelleen. Sovelluksessa myös toteutuu vaatimuksissa mainittu työpöytäajattelu. Sovellukset kuuluvat projektin vaiheiden mukaisille työpöydille, jolloin näkyvillä on ainoastaan kulloisessakin vaiheessa tarvittavat sovellukset ja ikkunat.

2.3 Tekniset vaatimukset

Sovellus toimii Linuxissa X Window System -käyttöympäristössä, jossa se myös on kehitetty. Projektin puitteissa ei jäänyt aikaa tehdä tarvittavia lisäyksiä, jotta sovellus olisi suoraan siirrettävissä Windowsiin, mutta siirto on jatkossa tehtävissä pienin muutoksin.

Verstaan ja alisovellusten välisessä kommunikaatiossa käytetään alkuperäisen suunnitelman mukaisesti TCP/IP -protokollaa. Tiedonsiirto on myöhemmin helposti salattavissa tekemällä tarvittavat muutokset `Communicator`-yksikköön.

2.4 Toteuttamatonta

Rajapintaan kuluvaa valikoiden rekisteröimiseen liittyvää toiminnallisuutta ei ehditty toteuttamaan. Komennon `menu.register` lähettäminen Verstaalle ei siis lisää valikkoa Verstaaseen. Tämä on tarvittaessa helposti toteutettavissa jatkokehitysvaiheessa.

3 Kommentointi- ja nimeämiskäytäntö

Tässä luvussa kuvataan sovellusta ohjelmoitaessa noudatettua kommentointi- ja nimeämiskäytäntöä.

Ohjelmakoodi on kommentoitu englanninkielisenä. Lähdekooditiedoston alussa mainitaan sen tarkoitus, projekti, haltija ja päiväys. Esimerkki 1 on malli projektin tuottamien kooditiedostojen alussa noudatetusta kommentointikäytännöstä.

Esimerkki 1: Kommentit kooditiedoston alussa

```
File: main.pas
Unit: main
Author: Verstas Project 2002:
 Antti Hakala, anthakal@cc.jyu.fi,
 Tomi Laamanen, laamanen@cc.jyu.fi,
 Paavo Nieminen, nieminen@cc.jyu.fi,
 Jukka Toivanen, jitoivan@cc.jyu.fi
Last Modified: 2002-11-27
Description:
 Includes Verstas Solver main form, TCP client thread and
 console application thread.
Notes:
 For Windows support all Libc related should be replaced with
 ShellAPI or so, and of course the corresponding methods like
 ExecApp should be redefined. A conditional define (like CLX)
 could be used with {$ifdef CLX} ... {$else} ... {$endif}.
```

Lisäksi kunkin aliohjelman toiminta, parametrit ja paluuarvot on selitetty lyhyesti.

Muuttujien nimeämisessä käytetään taulukon 1 mukaista käytäntöä. Metodit ja muuttujat on nimetty siten, että niiden tarkoitus ja toiminta on mahdollisimman pitkälle pääteltävissä niiden nimistä. Parametrien nimet on pyritty pitämään mahdollisimman yksinkertaisina, kuitenkin siten, että yhdistettynä metodin nimeen ne voidaan tulkita yksikäsitteisesti.

Tyyppi	Nimeämiskäytäntö	Esimerkki
Luokat	Ensimmäinen kirjain T, sanat isolla alkukirjaimella	`TMokkula, TProjectManager`
Rajapinta-määrittelyt	Sanat isolla alkukirjaimella	`Application, ProjectManager`
Metodit	Sanat isolla alkukirjaimella, ei alaviivaa	`GetGeometry, SetWindowPosition`
Parametrit	Pieni alkukirjain, ei alaviivaa, sanat isolla alkukirjaimella	`height, fileName`

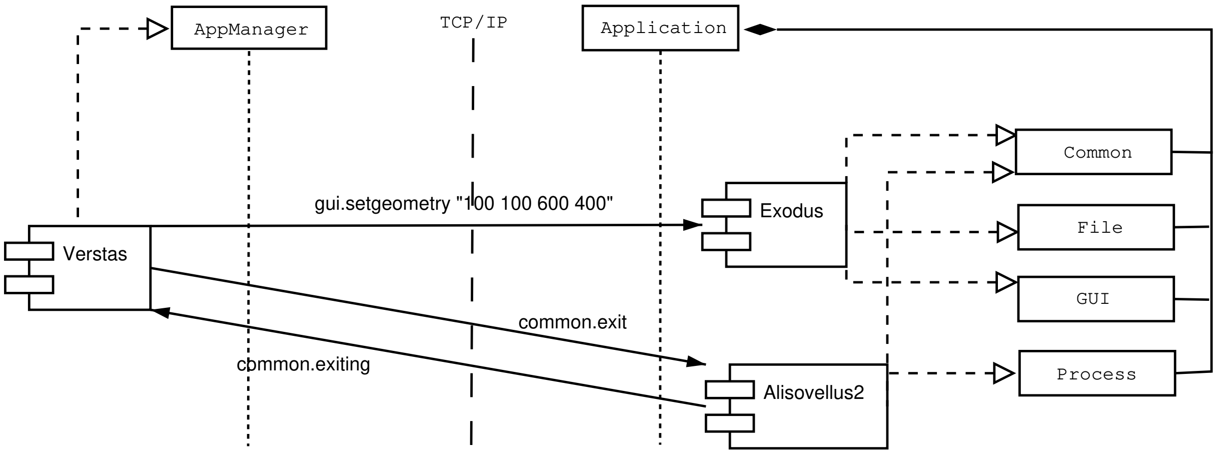
Taulukko 1: Nimeämiskäytäntö

4 Rajapinnat

Rajapintojen määrittely jakautuu kahteen osaan: `Application` ja `AppManager`. `Application`-rajapinta on rajapinta joka alisovelluksen on toteutettava, jotta se on se liitettävissä Verstaaseen. `AppManager` on rajapinta, jonka Verstas toteuttaa ja jonka komentoja käyttäen alisovellus ilmoittaa tapahtumistaan.

`Application`-rajapinta jakautuu alirajapintoihin `common`, `file`, `gui`, `console` ja `process`. Alisovellus ilmoittaa käynnistyessään (`AppManager`-rajapinnan `common.starting`-komennolla) mitkä alirajapinnat se toteuttaa. Näin Verstas saa tiedon mm. siitä onko alisovelluksella graafista käyttöliittymää, joka sen on otettava huomioon työpöytää järjestettäessä.

Käytännössä alisovellus ottaa käynnistyessään TCP/IP-yhteyden Verstaaseen ja rajapintojen komennot välitetään tekstimuodossa Verstaan ja alisovelluksen välillä.



Kuva 1: Rajapinnat ja ajonaikainen toiminta

4.1 Application

Toteuttaakseen `Application`-rajapinnan alisovelluksen tulee toteuttaa:

1. **Komentoriviparametri**t:
   `-iconic` Alisovellus käynnistyy pienennettynä.

   Lisäksi alisovellus pystyy käynnistettäessä asettamaan ikkunoidensa geometrian niillä komentoriviparametreillä jotka alisovellus antaa saadessaan komennon `common.getargs`. Ikkunat palautuvat samaan tilaan kuin missä ne olivat alisovelluksen saadessa komennon.

2. **Luvuissa 1.1-1.5 esitellyistä rajapinnoista vähintään Common-rajapinta.**

Rajapinnat on esitelty seuraavasti:

`komento “parametri1” “parametri2”...`	toimenpide

Komennon saadessaan alisovelluksen on toimittava toimenpiteen mukaan. Jos alisovellus toteuttaa rajapinnan, on sen toteutettava rajapinnan kaikki komennot ja täytettävä ilmoitusvelvollisuudet. Ilmoitukset on tehtävä mainitulla `AppManager`-rajapinnan ilmoituksella (ilmoitettu hakasuluissa).

4.1.1 Common

Komennot:

`common.exit`	Alisovellus sulkeutuu
`common.forcedexit`	Alisovellus sulkeutuu kysymättä käyttäjältä mitään.
`common.getargs`	Alisovellus palauttaa komentoriviparametrien muodossa tietoja, joiden avulla alisovellus pystyy käynnistyessä palauttamaan geometriansa (kts. Esim. 1, kohdat 5-6).
`common.ping`	Kysely alisovellukselta, alisovelluksen tulee vastata `common.pong`.
`common.pong`	(Verstaan vastaus alisovelluksen `common.ping`:iin. Ei vaadi toimenpiteitä alisovellukselta)

Ilmoitusvelvollisuudet:

- Alisovellus käynnistyy [`common.starting`]
- Alisovellus sulkeutuu [`common.exiting`]

4.1.2 File

`file.open ‘‘filename’’`	Alisovellus avaa tiedoston.
`file.save ‘‘filename’’`	Alisovellus tallentaa tiedoston.
`file.close ‘‘filename’’`	Alisovellus sulkee tiedoston.

Huomioita:

- Verstas olettaa ettei alisovelluksessa voi olla sama tiedosto auki monesti.
- Jos alisovelluksessa tallennetaan tiedosto “vanha” nimelle “uusi” (SaveAs): Alisovellus ilmoittaa Verstaalle `file.closed ‘‘vanha’’`, `file.opened ‘‘uusi’’`, `file.saved ’’uusi’’`.
- Jos alisovelluksessa luodaan uusi tiedosto jota ei ole vielä talletettu, alisovelluksen tulee ilmoittaa `file.modified ‘‘untitled’’`. Jos tiedosto talletetaan, toimitaan yo. tavalla.

Ilmoitusvelvollisuudet:

- tiedoston avaaminen [`file.opened`]
- tiedoston tallentaminen [`file.saved`]
- tiedoston sulkeminen [`file.closed`]

- tiedoston muuttaminen [`file.modified`]

4.1.3 GUI

`gui.setbounds ''Width Height X Y''`	Alisovellus asettaa ikkunansa annetun alueen sisään. (`X`,`Y`) on alueen yläkulma, `Width` alueen leveys ja `Height` alueen korkeus pikseleissä.
`gui.iconify`	Alisovellus pienenee.
`gui.restore`	Alisovellus palautuu.

4.1.4 Process

`process.execute ''command''`	Alisovellus suorittaa merkkijonolla tunnistettavan komennon, jonka se on aikaisemmin ilmoittanut verstaalle.

4.2 AppManager

AppManager toteuttaa seuraavat rajapinnat:

4.2.1 Common

`common.starting ''name'' ''interface1'' ''interface2''...`	Alisovellus kertoo käynnistyneensä, ilmoittaa tunnisteensa ja rajapinnat jotka se toteuttaa. `Common`-rajapintaa ei ilmoiteta, koska se on kaikille alisovelluksille pakollinen.
`common.exiting`	Alisovellus kertoo sulkeutuvansa.
`common.ping`	Alisovellus "pingaa" Verstasta. Verstaan on vastattava komennolla `common.pong`.
`common.pong`	Alisovelluksen vastaus Verstaan komentoon `common.ping`.
`common.args`	Alisovellus antaa merkkijonon, joka voidaan käynnistettäessä liittää sen komentoriviparametreihin (kts. esim 1, kohdat 6 ja 11).

4.2.2 File

`file.opened ''filename''`	Alisovellus kertoo avanneensa tiedoston.
`file.saved ''filename''`	Alisovellus kertoo tallentaneensa tiedoston.
`file.closed ''filename''`	Alisovellus kertoo sulkeneensa tiedoston.
`file.modified ''filename''`	Alisovellus kertoo ettei tiedoston tallennettu versio vastaa alisovelluksessa olevaa.

4.2.3 Message

`message.error` “`errormessage`”	Alisovellus pyytää Verstasta näyttämään käyttäjälle virheilmoituksen.
`message.inform` “`message`”	Alisovellus pyytää Verstasta näyttämään käyttäjälle viestin.

4.2.4 Menu

`menu.register` “`menuitem`” “`command`”	Alisovellus lisää Verstaaseen pikavalinnan sekä kertoo komennon, jonka haluaa liittää siihen. Alisovelluksen on toteutettava `process`-rajapinta, jonka kautta ilmoitus valikon käytöstä tulee (`process.execute` “`command`”).
`menu.remove` “`menuitem`”	Alisovellus poistaa pikavalintansa Verstaasta.

Esim 1.

1. Käyttäjä käynnistää Exoduksen Verstaasta. Alisovellus käynnistyy ja lähettää verstaalle ilmoituksen `common.starting` “`exodus`” “`gui`” “`file`”.
2. Käyttäjä työskentelee jonkin aikaa alisovelluksella.
3. Verstas kokeilee vieläkö alisovellus kuuntelee ja antaa alisovellukselle komennon `common.ping`.
4. Alisovellus vastaa `common.pong`.
5. Käyttäjä tallentaa projektin, jolloin Verstas antaa alisovellukselle komennon `common.getargs`.
6. Alisovellus palauttaa `common.args` “`--args 100,100,400,200,100,300,200,80`” (tiedot ikkunoidensa sijainneista ja koista). Verstas tallentaa tiedon projektitiedostoon.
7. Käyttäjä sulkee Verstaan, jolloin Verstas antaa alisovellukselle komennon `common.exit`.
8. Alisovellus palauttaa `common.exiting` ja sulkeutuu.
9. Verstas sulkeutuu.
10. Käyttäjä käynnistää Verstaan ja avaa edellisen projektin.
11. Verstas käynnistää Exoduksen: `exodus -args 100,100,400,200,100,300,200,80`.
12. Exodus käynnistyy samaan tilaan kuin missä se oli saadessaan komennon `common.getargs`.

Esim. 2:

1. Alisovellus rekisteröi verstaaseen pikavalinnan komennolla: `menu.register` “`Run`” “`run_cmd`”.
2. Käyttäjä valitsee Verstaasta alisovelluksen pikavalinnan Run.
3. Verstas lähettää alisovellukselle komennon `process.execute` “`run_cmd`”.
4. Alisovellus suorittaa toimenpiteen joka oli tarkoitus yhdistää Run-pikavalintaan.

5 Tiedostoformaatit

Tässä luvussa esitellään Verstaan käyttämät tiedostoformaatit.

5.1 Projektitiedosto

Projektitiedosto (`.vpr`) esitellään esimerkin 2 avulla. Kolme pistettä (`...`) tarkoittaa poisjätettyä kohtaa, joka toistaa edellä selitettyä kaavaa. Tiedostossa voi olla kommentteja kiharasulkeiden `{}` sisällä. Kommentit eivät voi olla sisäkkäisiä.

Puolipisteellä '`;`' merkitään aina syöterivin loppua. Rivillä '`end;`' merkitään suuremman syötekokonaisuuden loppua. Projektitiedoston ensimmäisellä rivillä on oltava tiedostoformaatin versio. Varsinainen tieto on avain-arvo -parien joukoissa, jotka taas sijaitsevat lohkoissa (`Item`, `App`) ja lohkot osastoissa (`Items`, `Applications`, `Bindings`) esimerkin mukaisesti.

Esimerkki 2: Projektitiedosto

```
vpr_version="1.00";
  { Tiedostoformaattiesimerkki.
    Lainausmerkkien sisällä olevat merkit käsitellään
    merkkijonoarvona. Merkkijonoon voidaan sisällyttää
    lainausmerkki, mutta se on koodattava kirjoittamalla
    kaksi lainausmerkkiä peräkkäin, kuten
    "Esimerkkimerkkijono, jossa on ""lainausmerkkejä"""
  }
  Items;
    Item(0)="TProjectRoot";
      { Metodilta GetAsString saatu tuloste }
      ...
    end;
    Item(1)="TProjectPhase";
      { Metodilta GetAsString saatu tuloste }
      ...
    end;
    ...
  end;
  Applications;
    App(1)="App";
      handlename = "emacs";
        { nimi, jolla tunnistetaan alisovellus }
      argstring = "-geometry 80x40+671+100";
        { tämä merkkijono katenoidaan alisovelluksen argumenttilistaan }
      openfiles = "/home/nieminen/work/testidata/test8/kukkua.f90";
        { tiedostot, jotka tässä alisovelluksessa tulee avata.
          Merkkijonon formaatti on se joka saadaan Kylixin
          merkkijonolistan ominaisuudesta TStringList.Commastring }
    end;
    App(2)="App";
      ...
    end;
    ...
  end;
  Bindings;
    {...}
    PhaseOfApp(1) = 5; { tarkoittaa: alisovellus App(1) on vaiheessa, joka on
Item(5). }
    ParentOf(4) = 2; { tarkoittaa: Item(2) on puurakenteessa Item(4):n vanhempi.
}
  end;
```

5.2 Verstaan asetustiedostot

Verstaan kaikki asetustiedostot ovat käyttäjäkohtaisia. Verstaan pääasetustiedosto on `verstasrc` (esimerkki 3), jossa määritellään galleriahakemisto (`Gallerypath`), TCP-portti (`Port`) ja maksimiviive alisovellukselta tulevalle vasteelle (`Timeout`).

Esimerkki 3: Verstaan pääasetustiedosto `verstasrc`

```
[General]
Gallerypath=/usr/local/verstas/gallery

[TCP]
Port=5000
Timeout=999
```

Alisovellusten asetustiedostossa `appsrc` (esimerkki 4) määritellään alisovelluskohtaisesti alisovelluksen nimi (`caption`), sen tarjoamat palvelut (`services`), alisovelluksen suorittava komento (`cmd`) ja parametrit jotka sille annetaan aina käynnistyksen yhteydessä (`paramstr`) sekä mallinnusvaihe (`phase`), johon alisovellus kuuluu.

Esimerkki 4: Alisovellusten asetustiedosto `appsrc`

```
[genesis]
caption=Genesis
services="2D geometry"
cmd=genesis paramstr=
phase=Problem

[exodus]
caption=Exodus
services="3D Geometry"
cmd=exodus1.1
paramstr=
phase=Problem

[solver]
caption=Solver
services="Solving"
cmd=solver
paramstr=
phase=Solver

[emacs]
caption=Emacs
services="Text File"
cmd=xemacs
paramstr=-l /usr/local/verstas/app-interface.el
phase=Model

[plot]
caption=Plot
services="Visualization"
cmd=plot1.3
paramstr=
phase=Results
```

Tiedostotyyppien asetustiedostossa `ftyperc` (esimerkki 5) määritellään kullekin tiedostotyypille kuvaus (`desc`), tiedostotunniste (`pattern`), alisovellukset, joihin tiedostotyyppi voidaan yhdistää (`apps`, tiedosto käynnistetään listan ensimmäiseen alisovellukseen), mallinnusvaihe (`phase`), johon tiedosto kuuluu sekä automaattinen avaus alisovellukseen (`autoOpen`) ja automaattinen päivitys tiedoston muuttuessa (`autoUpdate`).

Esimerkki 5: Tiedostotyyppien asetustiedosto `ftyperc`

```
[Exodus1.1]
desc=3D geometry
pattern=.exd
apps="exodus"
phase=Geometry
autoOpen=true
autoUpdate=true

[Fortran]
desc=Fortran code
pattern=.f90
apps="emacs" "solver"
phase=Model
autoOpen=true
autoUpdate=true

[Makefile]
desc=Makefile
pattern=Makefile
apps="solver" "emacs"
phase=Solver
autoOpen=true
autoUpdate=true

[pot]
desc=Pot-file
pattern=.pot
apps="plot"
phase=Results
autoOpen=true
autoUpdate=true
```

5.3 Verstas Solver -sovelluksen asetustiedosto

Verstas Solver -sovelluksen asetustiedosto (esimerkki 6) on käyttäjäkohtainen ja siinä määritellään käytettävät kirjastot (`Libraries`), Make-ohjelman asetukset (`Make`) sekä TCP-isäntä (`Host`) ja -portti (`Port`).

Esimerkki 6: Verstas Solver -sovelluksen asetustiedosto `solverrc`

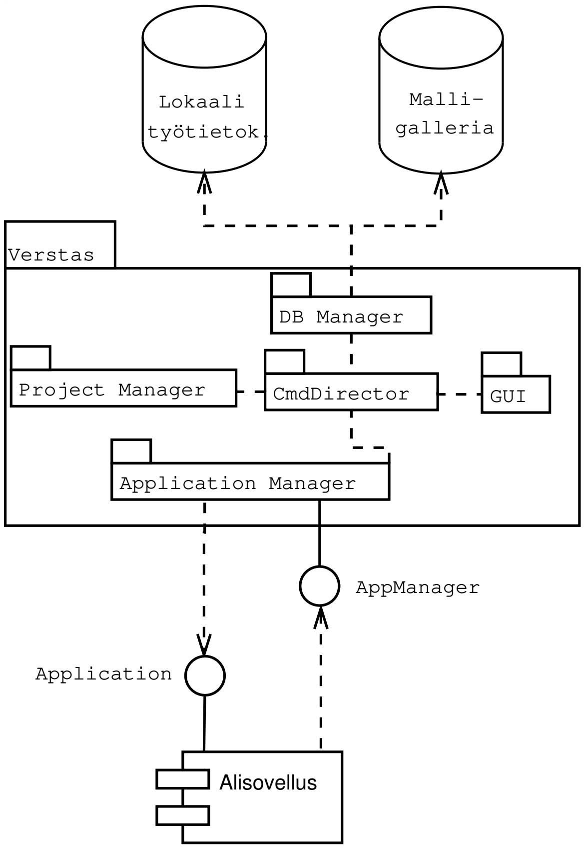
```
[Libraries]
Library1=nepsilon
Library2=nepsilong

[Make]
target1=all
target2=clean

[TCP]
Host=localhost
Port=5000
```

6 Verstas-arkkitehtuuri

Tässä luvussa kuvataan Verstas-sovelluksen arkkitehtuuria paketti- ja sekvenssikaavioin, sekä sovelluksen suunnittelussa käytettyjä suunnittelumaleja [3]. Luokat metodeineen on esitelty tarkemmin luvussa 7.

6.1 Yleisarkkitehtuuri

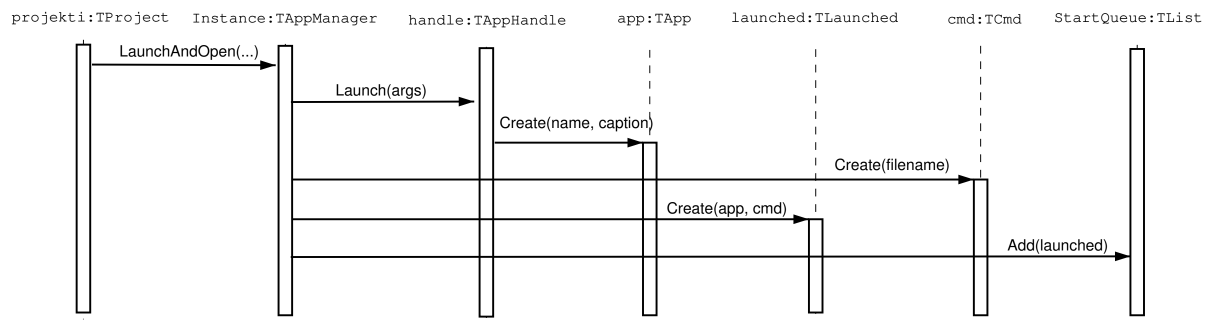
Kuvassa 2 on esitetty Verstaan arkkitehtuuria pakettikaavion muodossa. Verstas jakautuu osiin siten, että projektien hallinnasta, alisovellusten käynnistämisestä ja tietokantojen hallinnasta vastaa kustakin oma manageri. Managerien väliset ristiviittaukset on pyritty minimoimaan käyttämällä ns. Command- ja Mediator-malleja.



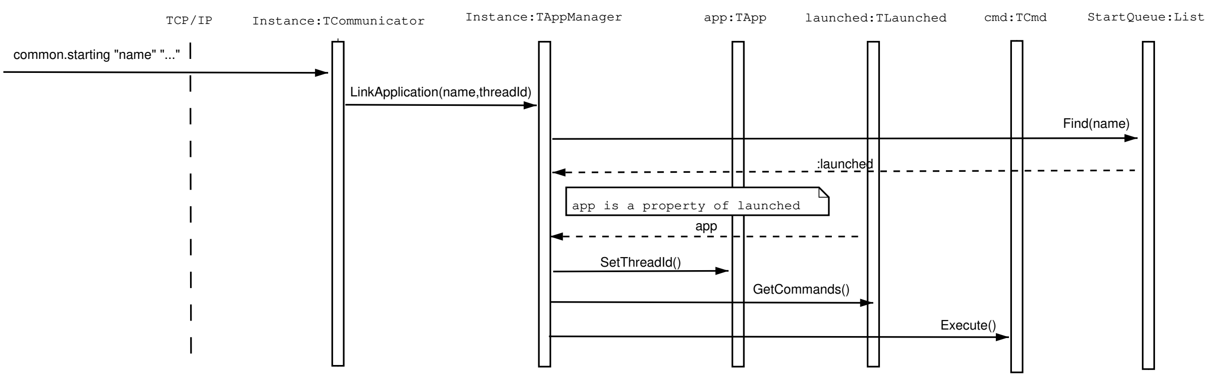
Kuva 2: Verstaan arkkitehtuuri pakettikaaviona

Kommunikointi alisovellusten kanssa on eristetty muusta Verstaasta siten, että viestien välityksen toteutustapa voidaan tarvittaessa vaihtaa. Kommunikointi hoidetaan TCP/IP:n kautta tekstimuodossa. Verstaan ja alisovellusten rajapinnat on esitelty luvussa 4.

Kuvassa 3 on esitetty alisovelluksen käynnistäminen `AppManagerin LaunchAndOpen`-metodin avulla. Manageri etsii tietorakenteestaan (`TList`) käynnistettävää sovellusta vastaavan `TAppHandle`-



Kuva 3: Alisovelluksen käynnistäminen



Kuva 4: Säietunnisteen linkitys `TApp`-olioon

tyyppisen kahvan. Kahvan `Launch`-metodi luo käynnistettävää sovellusta vastaavan `TApp`-tyyppisen olion, ja asettaa sille sovelluksen nimeä vastaavat tiedot.

Tämän jälkeen `AppManager` luo valmiiksi `TCmd`-tyyppiset komennot, jotka sovelluksen tulee suorittaa välittömästi, kun TCP-yhteys sovellukseen on muodostunut. Komennoista ja `TApp`-oliosta luodaan `TLaunched`-tyyppinen olio, joka talletetaan `StartQueue`-listaan (tyyppiä `TList)` odottamaan TCP-yhteyden muodostumista.

Yhteyden muodostuttua toimitaan kuvan 4 tavalla. `Kun common.starting`-viesti saapuu `Communicatorille`, se kutsuu vastaavan `VCmd`in avulla `AppManagerin LinkApplication`-metodia. Manageri etsii `StartQueue`-rakenteesta sovelluksen ilmoittamaa nimeä vastaavan `TLaunched`-olion, johon siis sisältyy valmiiksi luotu `TApp`-olio ja sovelluksen suoritettavaksi luodut komennot. `TApp`ille voidaan nyt asettaa säietunniste, jonka avulla se voidaan yksilöidä TCP-kommunikoinnissa, ja komennot voidaan suorittaa.

6.2 Command-malli

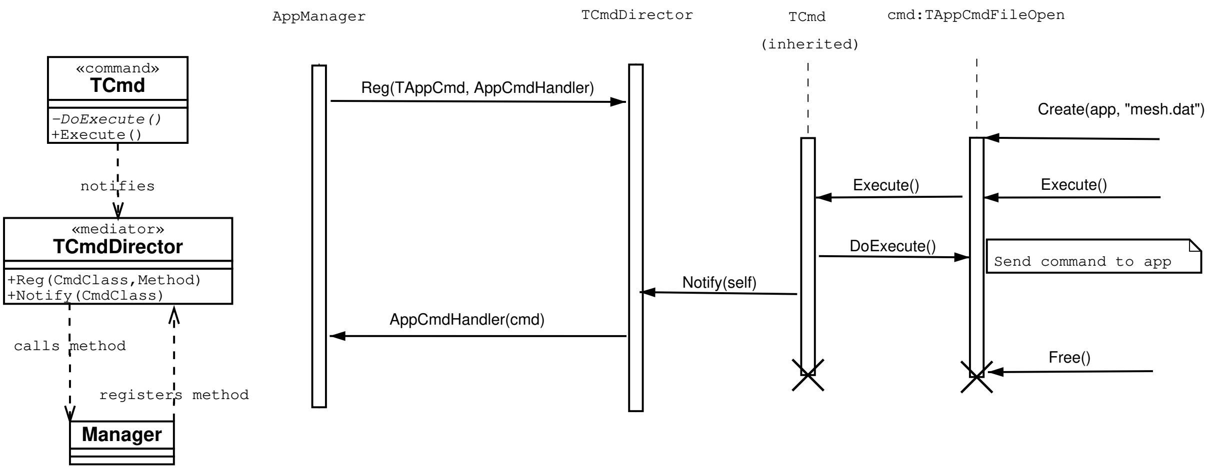
Command-mallin periaatteen mukaisesti lähes kaikkia Verstaan sisällä suoritettavia komentoja vastaa oma komentoluokka. Komentoluokasta voidaan luoda ilmentymiä milloin tahansa, mutta komennot suoritetaan vasta, kun niiden `Execute`-metodia kutsutaan.

Kaikki komennot on peritty abstaktista `TCmd`-luokasta, jossa `Execute`-metodi on määritelty. Metodi kutsuu ensin virtuaalista `DoExecute`-metodia, jonne perityt komennot sisällyttävät varsinaisen toiminnallisuutensa (tai joka edelleen kutsuu jotakin virtuaalista metodia, vrt. `TAppCmd` luvussa 7). Tämän jälkeen `Execute`-metodi kutsuu Mediator-mallin mukaista `TCmdDirector`-luokan instanssia, joka välittää tiedon komennon suorittamisesta kaikille kiinnostuneille.

6.3 Mediator-malli

Mediator-mallia mukaillen Verstaassa on singleton-tyyppinen luokka `TCmdDirector`, jonka tehtävänä on välittää tieto suoritetuista komennoista kaikille kyseistä tiedosta kiinnostuneille. Välitys hoidetaan siten, että viesteistä kiinnostuneet oliot rekisteröivät tietyn tyyppisen metodin kuuntelemaan tiettyä komentoluokkaa. Metodia kutsutaan, kun suoritetaan kyseisen tyyppinen tai siitä peritty komento. Metodille välitetään parametrina suoritettu komento-olio, jonka attribuutteja metodi voi käyttää hyödykseen. Katso lisätietoja kohdista 7.6,7.7 ja 7.30.

Kuvassa 5 on esimerkki tilanteesta, jossa `AppManager` rekisteröi kuuntelijan `TAppCmd`-luokasta perityille komennoille. Tämän jälkeen luodaan ja suoritetaan `TAppCmdFileOpen`-tyyppinen komento tiedostonavauskäskyn lähettämiseksi alisovellukselle. Komento suoritetaan kutsumalla sen `Execute`-metodia, joka edelleen kutsuu `DoExecute`-metodia. Siellä vastaava käsky lähetetään alisovellukselle. Lopuksi tieto komennon suorituksesta lähetetään mediatorin välityksellä `AppManagerille`.



Kuva 5: Kuuntelijan rekisteröiminen ja komennon suorittaminen

6.4 Singleton-malli

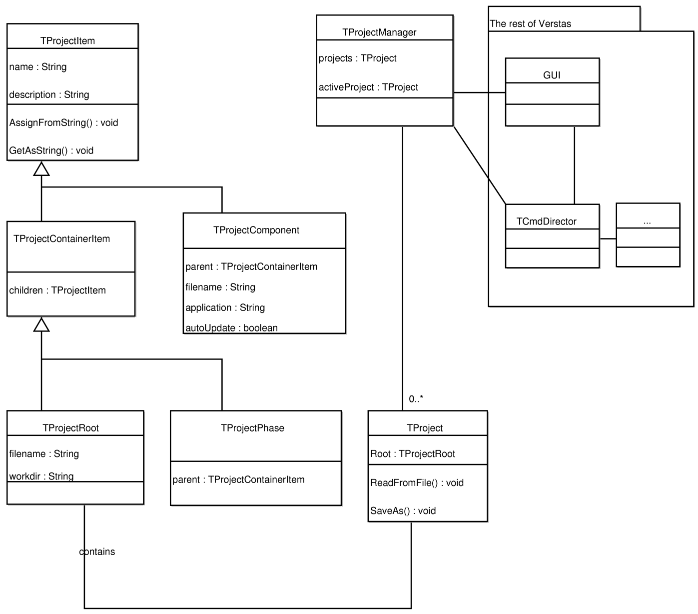
Sovelluksen toteutuksessa on käytetty runsaasti myös ns. singleton-mallia. Mallia käytetään, kun jostakin luokasta tarvitaan täsmälleen yksi instanssi. Tätä yksikäsitteistä instanssia käytetään kutsumalla erityistä funktiota, joka palauttaa viittauksen olioinstanssiin. Jos oliota ei ollut aikaisemmin luotu, se luodaan kutsun yhteydessä. Sen jälkeen olio on aina sama. Näin ei tule vahingossa tehtyä useita kopioita yksikäsitteiseksi tarkoitetusta oliosta.

6.5 Projektimanagerin tietorakenne

Mallinnusprojektia hallitaan sovelluksessa *projektimanagerin* avulla. *Projekti* on kokoelma tietoja, jotka liittyvät sovelluksella kulloinkin tehtävään simulointityöhön. Managerin toteutuksen pääasiat ilmenevät kaaviosta 6. Kaavio on karkea ja sen tarkoituksena on antaa yleiskuva toteutuksesta. Tarkemmat yksityiskohdat selviävät luvusta 7 sekä erityisesti ohjelmakoodin kommenteista.

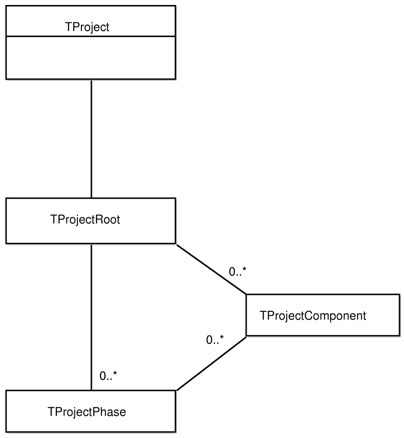
Projektimanageri sisältää listan avoimista projekteista, joista yksi kerrallaan voi olla aktiivinen. Kaikki muokkausoperaatiot on tarkoitus kohdistaa aina aktiiviseen projektiin - muut projektit voivat olla auki tietojen selaamista ja hakemista varten.

Kommunikointi muun sovelluksen kanssa tapahtuu periaatteessa yksinomaan viestinvälitysjärjestelmän ja käyttöliittymän avulla. Käytännössä sovellus jouduttiin toteuttamaan sellaisella kiireellä, että osa kommunikoinnista vuotaa viestinvälitysjärjestelmän ohitse. Ylläpitäjä voi tarpeelliseksi katsoessaan poistaa näitä rikkeitä aina havaitessaan niitä.



Kuva 6: Projektimanagerin luokat. Manageri voi sisältää useita projekteja, mutta vain yksi on kerrallaan aktiivinen. Kommunikointi muun sovelluksen kanssa tapahtuu ainakin periaatteessa yksinomaan Verstaan sisäisen komentojärjestelmän kautta.

Yksittäisen projektin tiedot tallennetaan sisäisesti puurakenteena, jonka sisältö ilmenee tarkemmin kaaviosta 7. Jokaista projektia vastaa täsmälleen yksi *juurisolmu*, joka sisältää projektikohtaisia tietoja. Juurisolmulla on lapsisolmuja, jotka voivat olla projektin *päävaiheita* tai *mallinnuskomponentteja*. Vaiheilla voi olla lapsina komponentteja sekä myös toisia vaiheita, vaikka tätä ei ole kaavioon merkitty.



Kuva 7: Projektin puurakenne: Projektilla on täsmälleen yksi juurisolmu. Juuri voi sisältää useita vaiheita ja komponentteja. Myös vaiheet voivat sisältää komponentteja

7 Sovelluksen luokat käännösyksiköittäin

Seuraavassa esitellään sovelluksen luokat käännösyksiköittäin. Luokista on esitetty yleinen kuvaus, jonka lisäksi tärkeimmät metodit on esitelty erikseen.

7.1 App.pas

Sisältää yhden alisovelluksen kapseloimiseen tarvittavat luokat.

`TFileInfo = class`

Apuluokka, joka pitää sisällään tietoa yhdestä tietyssä sovelluksessa avatusta tiedostosta. Tallennettaviin tietoihin kuuluu tiedoston nimi polkuineen sekä se, onko tiedostoa muokattu.

`TApp = class`

Luokka, joka Verstaan sisällä kuvaa yhtä ulkoista alisovellusta. Sovelluksesta tiedetään nimi, sen toteuttamat rajapinnat ja säietunniste, jonka perusteella sovellukselle voidaan lähettää viestejä `Communicator`in kautta. Sovellus myös pitää kirjaa siitä, mitä tiedostoja sillä on auki ja onko niitä muokattu edellisen tallennuksen jälkeen.

`function FindFileInfo(str:string) : integer;`

Palauttaa tiedon siitä, missä indeksissä parametrina annetun tiedoston muokkaustiedot (tyyppiä `TFileInfo`) ovat sovelluksen tietorakenteessa, tai -1 jos tietoja kyseisestä tiedostosta ei ole.

`function Implements(iface: string): boolean;`

Palauttaa true, jos sovellus toteuttaa parametrin annetun rajapinnat.

`procedure AddInterface(iface: string);`

Lisää oliolle tiedon siitä, että se toteuttaa parametrina annetun rajapinnan.

`procedure SetThreadId(id: Cardinal);`

Asettaa sovelluksen säietunnisteen. Metodi myös rekisteröi `CmdDirector`iin metodin, jolle lähetetään tieto `VCmdFileSaved`-komentojen suorittamisesta. Rekisteröinti tehdään täällä, koska viestin kuunteleminen on ajankohtaista vasta, kun olioa vastaavaan alisovellukseen on TCP-yhteys.

`function FileOpened(filename: string) : TFileInfo;`

Metodia kutsutaan, kun alisovellus on ilmoittanut avanneensa tiedoston. Metodi luo tiedostosta `TFileInfo`-tyyppisen rakenteen ja lisää sen omaan tietorakenteeseensa.

`procedure FileSaved(filename: string);`

Metodia kutsutaan, kun alisovellus on tallettanut tiedoston. Tallentaminen tarkoittaa, että tiedostossa ei enää ole tallentamatonta tietoa, joten metodi päivittää sovelluksen tiedostoa koskevat muokkaustiedot vastaavasti.

`procedure FileClosed(filename: string);`

Kutsutaan, kun alisovellus on sulkenut tiedoston, minkä jälkeen sitä koskevia muokkaustietoja ei enää talleteta.

`procedure FileModified(filename: string);`

Kutsutaan, kun alisovelluksessa on muokattu tiedostoa. Tieto muokkauksesta talletaan, ja tiedoston katsotaan sisältävän tallentamatonta tietoa siihen saakka, kunnes sovellus lähettää tiedon tiedoston tallentamisesta tai sulkemisesta.

`procedure GetModified(sl : TStringList);`

Palauttaa parametrina saamassaan listassa tiedon kaikista tiedostoista, joita on kutsumishetkellä muutettu muttei tallennettu.

`procedure GetOpened(sl : TStringList);`

Palauttaa parametrina saamassaa listassa tiedon kaikista tiedostoista, jotka sovelluksessa on auki.

`function IsOpen(filename: string) : boolean;`

Palauttaa tiedon siitä, onko parametrina annettu tiedosto auki alisovelluksessa.

`procedure VCmdFileSavedhandler(cmd:TCmd);`

Metodi, joka rekisteröidään `CmdHandler`iin kuuntelemaan `TVCmdFileSaved`-tyyppisten komentojen suorittamista. Suoritettu komento-olio saadaan parametrina, jolloin kommenolta saadaan tieto talletetusta tiedostosta. Jos sama tiedosto on auki tässä sovelluksessa, se ladataan uudestaan, mikäli tiedoston `autoUpdate`-ominaisuus on käytössä.

`property threadId: Cardinal read fthreadId write SetThreadId;`

Ominaisuuden avulla voidaan tutkia ja asettaa sovelluksen säietunniste.

`property name: string read fname write fname;`

Nimi, jolla sovellus yksilöidään `appsrc`- ja `ftyperc`-tiedostoissa.

`property caption: string read fcaption write fcaption;`

Sovelluksen name-tunnistetta kuvaavampi nimi, joka voidaan määrittää `appsrc`-tiedostossa. `Caption` on teksti, joka näkyy pääikkunan Windows-valikkossa sovelluksen ollessa käynnissä.

`property parent : TProjectContainerItem read fparent write fparent;`

Se mallinnusprojektin osa, jonka alaisuudessa sovellus toimii. Yleensä `parent`-ominaisuus on jokin projektin vaihe, johon sovellus liittyy, tai projektin juuri, jos sovellus ei liity mihinkään vaiheeseen.

`property args : string read fargs write fargs;`

Parametrit, jotka kuvaavat sovelluksen tilaa. Parametrit kysytään sovellukselta TCP:n kautta kun projektia aletaan tallentaa.

7.2 AppCmd.pas

Sisältää luokkina kaikki komennot, joita alisovelluksille voidaan lähettää.

`TAppCmd = class(TCmd)`

Kaikkien alisovelluksille lähetettävien komentojen kantaluokka.

`procedure AppDoExecute; virtual; abstract;`

Abstrakti metodi, joka kaikkien `TAppCmd` -luokasta perittyjen konkreettisten luokkien tulee toteuttaa. `DoExecute`-metodi kutsuu `AppDoExecute`-metodia, joka sisältää komennon varsinaisen toiminnallisuuden. Kyseessä on ns. template-metodi (kts. [3]).

`procedure SendMsg(msg: string);`

Lähettää tekstimuotoisen komennon alisovellukselle. Kutsuu `TCommunicator`in vastaavaa metodia.

`constructor Create(a: TApp; intrface: string);`

Luokan konstruktori. Parametrina annetaan alisovellus, jolle komento tulee lähettää, sekä rajapinta, johon komento kuuluu. Konstruktoria ei kutsuta suoraan, vaan sitä kutsuvat `TAppCmd`-luokasta perityt oliot konstruktorissaan.

`procedure DoExecute; override;`

Sisältää kaiken sellaisen toiminnallisuuden, joka on yhteistä kaikille alisovelluksille lähetettäville komennoille. Kutsuu edelleen virtuaalista `AppDoExecute`-metodia.

`function GetInterface: string;`

Palauttaa tekstimuodossa rajapinnan, johon komento kuuluu.

7.2.1 Rajapintakohtaiset kantaluokat

Kaikilla rajapinnan eri osiin (kts. kohta 4) kuuluvilla komennoilla on oma kantaluokkansa, joista kyseiset komennot peritään. Luokat ovat abstrakteja, eikä niillä ole konstruktorin lisäksi muita metodeja. Seuraavassa luetellaan nämä kantaluokat.

`TAppCmdCommon = class(TAppCmd)`

Abstrakti kantaluokka kaikille `common`-rajapintaan kuuluville komennoille.

`TAppCmdFile = class(TAppCmd)`

Abstrakti kantaluokka kaikille `file`-rajapintaan kuuluville komennoille.

`TAppCmdProcess = class(TAppCmd)`

Abstrakti kantaluokka kaikille process-rajapintaan kuuluville komennoille.

`TAppCmdMenu = class(TAppCmd)`

Abstrakti kantaluokka kaikille menu-rajapintaan kuuluville komennoille.

`TAppCmdGUI = class(TAppCmd)`

Abstrakti kantaluokka kaikille `common`-rajapintaan kuuluville komennoille.

7.2.2 Konkreettiset komentoluokat

Seuraavassa luetellaan kaikki alisovelluksille lähetettäviä komentoja vastaavat luokat. Nämä luokat ovat konkreettisia, ja niihin kuuluu konstruktorin lisäksi `AppDoExecute`-metodi. Konstruktorien parametrina on aina sovellus, jolle komento lähetetään, eikä sitä ole seuraavassa erikseen mainittu.

`TAppCmdCommonExit = class(TAppCmdCommon)`

Alisovelluksen sulkemiskomento.

`TAppCmdCommonForcedExit = class(TAppCmdCommon)`

Alisovelluksen pakotettu sulkemiskomento.

`TAppCmdCommonGetArgs = class(TAppCmdCommon)`

Käsky alisovellukselle lähettää tila- ja geometriatietonsa tallentamista varten.

`TAppCmdCommonPing = class(TAppCmdCommon)`

Komento pingin lähettämiseksi alisovellukselle.

`TAppCmdCommonPong = class(TAppCmdCommon)`

Komento pong-vastauksen lähettämiseksi alisovelluksen pingiin vastaamiseksi.

`TAppCmdFileOpen = class(TAppCmdFile)`

Käsky avata tiedosto. Konstruktorin parametrina sovelluksen lisäksi tiedoston nimi polkuineen.

`TAppCmdFileSave = class(TAppCmdFile)`

Käsky tallentaa tiedosto. Konstruktorin parametrina myös tallennettavan tiedoston nimi

`TAppCmdFileClose = class(TAppCmdFile)`

Käsky sulkea tiedosto. Tiedoston nimi annetaan konstruktorin parametrina.

`TAppCmdProcessExecute = class(TAppCmdProcess)`

Käsky suorittaa konstruktorin parametrina annettu komento, jonka alisovellus on aiemmin rekisteröinyt.

`TAppCmdGUIIconify = class(TAppCmdGUI)`

Käsky minimoida kaikki graafiset ikkunat.

`TAppCmdGUIRestore = class(TAppCmdGUI)`

Käsky palauttaa kaikki mimoidut ikkunat.

`TAppCmdGUISetBounds = class(TAppCmdGUI)`

Alisovellukselle annettava komento asettaa itsensä annettujen rajojen sisään.

`constructor Create(a: TApp; X, Y, width, height: integer);`

Konstruktorille annetaan alueen x- ja y-koordinaatit sekä leveys ja korkeus.

7.3 AppHandle.pas

Sisältää alisovelluksen käynnistämiseen tarvittavat luokat.

`TAppHandle = class`

Luokka on eräänlainen kahva yhteen käynnistettävissä olevaan alisovellukseen. Kahvan avulla voidaan saada tietoja kahvan edustamasta sovelluksesta, luoda sovellusta vastaava `TApp`-tyyppinen olio ja asettaa sen oletustiedot, sekä käynnistää varsinainen sovellus.

Erityisesti kahva myös tietää alisovellukset tarjoamat palvelut, jotka luetellaan käyttäjälle Verstaan valikossa. Käyttäjän valitessa tietyn palvelun käynnistetään sen tarjoava sovellus.

`procedure AddService(service: string);`

Lisää tiedon siitä, että kahvan edustama sovellus tarjoaa myös parametrina annetun palvelun.

`procedure GetServices(var list: TStringList);`

Palauttaa parametrina saamassaan listassa kaikki kahvan edustaman sovelluksen tarjomat palvelut.

`property tag: integer read ftag write ftag;`

Yksikäsitteinen tunniste, jonka perusteella kahva yksilöidään mm. Verstaan valikoissa.

`property name: string read fname write fname;`

Kahvan edustaman sovelluksen nimi. Asetaan luodun `TApp`-tyyppiselle olion nimeksi kun sovellus käynnistetään.

`property appCaption : string read fappCaption write fappCaption;`

Sovelluksen kuvaavampi nimi. Asetetaaan vastaavan `TApp`-olion caption-ominaisuudeksi, kun sovellusta käynnistetään.

`property defaultPhase : string read fdefaultPhase write fdefaultPhase;`

Kahvan edustaman sovelluksen oletusvaihe tekstimuodossa. Vaiheesta tulee vastaavan `TApp`-olion `parent`-ominaisuus, mikäli aktiivisessa projektissa on kyseinen vaihe. Mikäli vaihetta ei ole, sovelluksen parentiksi tulee projektin juuri.

`property launchCommand : string read flaunchCommand write flaunchCommand;`

Konkreettinen komento, jonka avulla kahvan edustama sovellus voidaan käynnistää.

`property launchParameters : string read flaunchParameters write flaunchParameters;`

Parametrit, jotka sovellukselle annetaan aina käynnistyksen yhteydessä.

`function GetApp : TApp;`

Palauttaa kahvan edustamaa sovellusta vastaavan `TApp`-tyyppisen olion.

`function Launch(args: string='') : TApp;`

Käynnistää kahvan edustaman sovelluksen ja palauttaa vastaavan `TApp`-olion. Parametrina voidaan antaa sellaiset parametrit, jotka sovelluksella annetaan `launchParameters`-ominaisuudessa mainittujen parametrien lisäksi.

7.4 AppManager.pas

Sisältää luokat alisovellusten hallintaan.

`TLaunched = class`

Apuluokka, johon talletetaan käynnistetyn sovelluksen tiedot siihen saakka, kunnes se muodostaa yhteyden Verstaaseen.

`constructor Create(app: TApp; cmds: TList);`

Konstruktorin parametrina annetaan jo luotu sovellusta kuvaava olio, sekä lista kommennoista, jotka sovelluksen täytyy suorittaa heti, kun siihen saadaan yhteys.

`property app: TApp read fapp write fapp;`

Ominaisuus, jonka avulla saadaan selville `TApp` -olio, joka vastaa käynnistettyä sovellusta.

`procedure ExecuteCmds;`

Suorittaa kaikki konstruktorin parametrina annetulla listalle olleet komennot. Kutsutaan, kun alisovellukseen on saatu yhteys.

`TAppManager = class`

Alisovellusten hallinnasta vastaava manageri.

`procedure AddToStartQueue(app: TApp; cmds: TList);`

Lisää sisäiselle listalle tiedon siitä, että parametrina annettua oliota vastaava sovellus on käynnistetty. Toisena parametrina kommenot, jotka sovelluksen tulisi suorittaa, kun yhteys siihen on muodostettu. Luo tiedoista `TLaunched`-tyyppisen olion.

`procedure GetLaunchables(var list: TList);`

Palauttaa saamassaan listassa merkkijonomuodossa kaikki sovellukset, jotka Verstas voi käynnistää.

`procedure GetFileTypes(var list: TList);`

Palauttaa saamassaan listassa merkkijonomuodossa kaikki Verstaan tuntemat tiedostotyypit.

`procedure AddLaunchable(handle: TAppHandle);`

Lisää parametrina annetun kahvan sisäiseen rakenteeseen. Tämän jälkeen manageri olettaa, että kahvaa vastaava sovellus voidaan käynnistää.

`procedure AddFileType(ftype: TFileType);`

Lisää parametrina annetun tiedostotyypin sisäiseen rakenteeseen.

`procedure LaunchApp(tag: integer); overload;`

Käynnistää sovelluksen kokonaislukutunnisteen perusteella.

`procedure LaunchApp(handle: TAppHandle); overload;`

Käynnistää annettua kahvaa vastaavan sovelluksen.

`procedure LaunchAndOpen(name: string; args: string; files: string; parent: TProjectContainerItem);`

Käynnistää nimen perusteella annetun sovelluksen annetuilla argumenteilla, avaa siihen `TStrings`-tyypin `commastring`-muodossa annetut tiedostot ja asettaa annetun olion luodun `TApp`-olion `parent`-ominaisuukseksi.

`function LinkApplication(name: string; threadId: Cardinal): TApp;`

Kutsutaan, kun sovellus ilmoittaa Verstaalle käynnistyneenesä. Etsii annetun sovelluksen nimen perusteella vastaavan `TLaunched`-olion käynnistysjonosta, ja asettaa sen osoittaman `TApp`-olion säietunnisteen. Suoritaa myös `TLaunched`-olion sisältämät komennot.

`function GetInstance(threadId: Cardinal): TApp; overload;`

Palauttaa vastaavan `TApp`-olion säietunnisteen perusteella.

`function GetInstance(name: string): TApp; overload;`

Palauttaa vastaavan `TApp`-olion sovelluksen nimen perusteella.

`procedure GetInstances(list: TList);`

Palauttaa listassa kaikki käynnissä olevat sovellukset.

`function GetNumInstances : integer;`

Palauttaa käynnissä olevien sovellusten lukumäärän.

`procedure DeleteInstance(threadId: Cardinal);`

Poistaa `TApp`-olion säietunnisteen perusteella.

`function GetFileType(fileName: string) : TFileType;`

Palauttaa annettua tiedoston nimeä vastaavan `TFileType`-olion, mikäli sellainen on olemassa.

`function GetAppHandle(app: string) : TAppHandle; overload;`

Palauttaa nimen perusteella kahvan sovellukseen.

`function GetAppHandle(tag: integer) : TAppHandle; overload;`

Palauttaa kokonaislukutunnisteen perusteella kahvan sovellukseen.

`function GetAppByFile(fileName: string) : TApp;`

Palauttaa `TApp`-olion, jota vastaavassa sovelluksessa on auki annettu tiedosto.

7.5 AppWnd.pas

Sisältää luokat, jotka kuvaavat alisovelluksen graafista ikkunaa. `TAppWnd`-luokka periytyy yleisemmästä ikkunaluokasta `TWnd`, ja se toteuttaa saamansa geometriakomennot lähettämällä komennon omalle alisovellukselleen.

7.6 Cmd.pas

Sisältää abstraktin `TCmd`-luokan.

`TCmd = class`

`procedure Execute;`

Metodi, jolla komento suoritetaan. Kutsuu virtuaalista `DoExecute`-metodia, joka on toteutettu kaikissa suoritettavissa komennoissa. Tämän jälkeen metodi kutsuu `CmdDirector`in `Notify`:ta, joka välittää tiedon komennon suorittamisesta kaikille asiasta kiinnostuneille.

`procedure DoExecute; virtual; abstract;`

Abstrakti metodi, jota `Execute` kutsuu. `TCmd`-luokasta perityt luokat voivat sisällyttää `DoExecute`-metodiin omaa toiminnallisuuttaan.

7.7 CmdDirector.pas

Sisältää viestien välitykseen tarvittavat luokat.

`TListenMethod = procedure (cmd: TCmd) of object;`

Määrittää tyypin sellaisille metodeille, jotka ovat rekisteröitävissä kuuntelemaan komentojen suoritusilmoituksia.

`TListenCmd = class of TCmd`

Määrittää kuunneltavien komentojen perustyypin. Kuunnella voidaan siis kaikkia `TCmd`-luokasta perittyjä komentoja.

`TListener = class`

Apuluokka, joka sisältää tiedot kuunneltavan komennosta (tyyppiä `TListenCmd`) sekä metodista, jota kutsutaan kun komento on suoritettu (tyyppiä `TListenMethod`).

`TCmdDirector = class`

Varsinainen komentojen välittämisestä vastaava singleton-luokka.

`function Reg(cmd: TListenCmd; method: TListenMethod): TListener;`

`Reg`-funktion avulla voidaan rekisteröidä metodi kuuntelemaan tietyn tyyppisiä komentoja. Annettua metodia siis kutsutaan aina, kun suoritetaan komento, joka on annettua tyyppiä tai on peritty siitä.

Esimerkki: jos rekisteröidään metodi kuuntemaan `TAppCmd`-tyyppisiä komentoja, kutsutaan sitä myös, kun suoritetaan `TAppCmd`:sta peritty `TAppCmdFileOpen`-komento.

Funktio palauttaa luodun `TListener`-tyyppisen olion, jonka avulla kuuntelija voidaan myöhemmin poistaa.

`procedure UnReg(listener: TListener);`

Poistaa rekisteröidyn kuuntelijan. Parametrina on annettava olio, jonka `Reg`-funktio rekisteröinnin yhteydessä palautti.

`procedure Notify(cmd:TCmd);`

Metodi, jota `TCmd`-olio kutsuu, kun komennon suoritus on päättynyt. Tämän jälkeen tieto suorituksesta lähetetään kaikille metodeille jotka ovat rekisteröityneet kuuntelemaan kyseisen tyyppisiä komentoja.

7.8 CmdParser.pas

Sisältää luokat Verstaan vastaanottamien komentojen käsittelyyn. Yksikön vastuulle kuuluu komentojen jäsentäminen, eli komento-osan ja parametrien irrottaminen, sekä Verstaan vastaavien käsittelykomentojen kutsuminen. Rajapinnan muuttaminen tai laajentaminen vaatii `CmdParser`- ja `AppCmd` -yksiköiden muuttamista.

`TWaited = class`

Apuluokka, jonka esiintymä luodaan, kun tietyltä sovellukselta aletaan odottaa tiettyä viestiä.

`constructor Create(iThreadId: Cardinal; iMsg: string);`

Konstruktorin parametrina annetaan odotettavan sovelluksen säietunniste sekä odotettava viesti.

`property msg: string read fmsg;`

Odotettava viesti.

`property threadId : Cardinal read fthreadId;`

Sen sovelluksen säietunniste, jolta viestiä odotetaan.

`received: boolean;`

Asetetaan todeksi, kun odotettu viesti on saapunut.

`TCmdParser = class`

Luokka, jota `Communicator` käyttää vastaanotetun komennon jäsentämiseen. `AppManager`-rajapintaan tulevat muutokset aiheuttavat muutoksia ainoastaan tässä luokassa.

`procedure ParseCmd(txt: string; threadId: Cardinal);`

Erottelee saamastaan tekstistä komento-osan ja lainausmerkein (“) erotetut parametrit, ja kutsuu `ExecuteCmd`-metodia.

`procedure ExecuteCmd(command: string; params: array of string);`

Käsittelee Verstaan vastaanottaman komennon. Käytännössä luo ja suorittaa vastaavat `TVCmd`-tyyppiset komennot. Parametrina annetaan vastaanotettu komento tekstimuodossa sekä taulukossa komennolla olleet parametrit.

`function WaitFor(AppThread: Cardinal; msg: string) : boolean;`

Odottaa, kunnes annettua säietunnistetta vastaavalta sovelukselta on tullut annettu viesti. Palauttaa tosi, mikäli viesti saapuu ennen `Communicatorin subAppTimeout`-ominaisuuden määräämä aika on kulunut.

Muut odottamisen aikana saapuneet viestit käsitellään normaalisti, eli metodia käytettäessä on pidettävä mielessä mahdolliset synkronointiongelmat (kts. Ideoita jatkokehitäjälle).

`procedure ResetNumArgs;`

Nollaa `numArgs`-laskurin. Kutsutaan, kun projektia aletaan tallettaa.

`property numArgs : integer read fnumArgs;`

Laskuri, joka pitää kirjaa sovelluksilta tulleista common.args-tyyppisistä viesteistä. Käytetään projektia talletettaessa jotta tiedettäisiin, joko kaikki alisovellukset ovat palauttaneet tilatietonsa.

7.9 Communicator.pas

Sisältää TCP-kommunikointiin liittyvät luokat. `Communicatorin` avulla voidaan käynnistää TCP-palvelin kuuntelemaan valittua porttia. Vastaanotettujen viestin tulkitsemiseen `Communicator` käyttää `TCmdparser`-singletonia. Asiakkaat (eli eri alisovellukset) tunnistetaan niille käynnistettyjen säikeiden threadId -ominaisuuksien avulla, joten `Communicator` ei tarvitse mitään tietoa siitä, minkälaisia sovelluksia se palvelee.

`TParseable = class`

Apuluokka, johon talletetaan vastaanotetut komennot, kunnes ne jäsennetään. Jäsentäminen tapahtuu synkronoidusti, minkä vuoksi vastaanotetut komennot talletetaan jonorakenteeseen. Komennot jäsennetään saapumisjärjestyksessä.

`parsestr: string`

Saapunut komento ja parametrit yhtenä merkkijonona.

`threadId: Cardinal`

Säikeen tunniste, jonka avulla viestin lähettänyt alisovellus tunnistetaan komennon jäsentämisvaiheessa.

`TCommunicator = class`

Verstaan TCP-palvelin, joka palvelee kaikkia asiakkaita yhdessä portissa. Asiakkaille luodaan oma säie, jonka perusteella niiltä tulleet viestit tunnistetaan Verstaan sisällä.

`procedure TcpServerDisconnect(AThread: TIdPeerThread);`

Metodi, jota palvelinkomponentti kutsuu, kun yhteys johonkin asiakkaaseen on sulkeutunut. Säilöö asiakkaan säietunnisteen jonoon, ja kutsuu synkronoidusti `doDisconnect`-metodia.

`procedure doDisconnect;`

Luo ja suorittaa komennon asiakasta vastaavan `TApp`- ja mahdollisen `TWnd` -olion poistamiseksi Verstaasta.

`procedure TcpServerExecute(AThread: TIdPeerThread);`

Metodi, jossa luetaan asiakkailta tulevia viestejä. Luo `TParseable`-tyyppisen olion, tallettaa sen jäsennysjonoon ja kutsuu synkronoidusti `doParse`-metodia.

`procedure doParse;`

Poistaa ensimmäisenä saapuneen viestin jäsennysjonosta, ja kutsuu `CmdParseria` viestin jäsentämiseksi.

`function Send(thread: TIdPeerThread; msg: string) : boolean;`

Lähettää annetun asiakkaalle, jonka säie on annettu paremetrina. Palauttaa tosi, jos lähetys onnistuu.

`function SendMsg(msg: string; threadId: Cardinal) : boolean;`

Lähettää annetun viesti asiakkaalle, jonka säietunniste on annettu parametrina. Palautta tosi, jos lähetys onnistuu.

`procedure SendAll(st: string);`

Lähettää annetun viesti kaikille asikkaille.

`procedure Disconnect(threadId: Cardinal);`

Katkaisee yhteyden asiakkaaseen, jonka säietunniste annetaan parametrina.

`procedure StartServer;`

Käynnistää palvelimen. Yrittää asettua kuuntelemaan porttia, joka on määritetty `defaultPort`-ominaisuudessa. Mikäli ei onnistu, yrittää aina seuraavaksi suurempaa porttia kunnes kuunteleminen onnistuu. Kuunneltava portti asetetaan `verstasPort`-ominaisuuden arvoksi.

`property defaultPort: integer read fdefaultPort write fdefaultPort;`

Verstaan oletusportti, jota palvelinta käynnistettäessä ensimmäisenä pyritään kuuntelemaan.

`property verstasPort: integer read vport;`

Ominaisuus, joka kertoo mitä porttia Verstaan tämä ilmentymä kuuntelee. Arvo on 0 mikäli palvelinta ei ole käynnistetty.

`property subAppTimeout: integer read timeout write timeout;`

Aika, joka alisovellukselta odotetaan vastausta viesteihin, ennen kuin yhtes todetaan katkeneeksi.

7.10 DBManager.pas

Sisältää luokat useiden tietokantojen yhtäaikaiseen hallintaan.

`type TDBManager = class`

Luokka, joka sisältää käytettävät tietokannat ja tiedon aktiivisesta tietokannasta. Tällä hetkellä toteutus sisältää vain yhden tietokannan (`TFSDatabase`). Myöhemmin jos tietokantoja tarvitaan enemmän, voidaan lisätä tietorakenne joka sisältää tietokannat sekä komennot tietokannan valitsemiseen jne.

`class function Instance: TDBManager;`

Singleton-patternin mukainen metodi, jolla saadaan viittaus tietokantamanagerin instanssiin.

`procedure SetCurrentDB(db: TFSDataBase);`

Asettaa aktiivisen tietokannan.

`procedure Init();`

Alustustoiminto, joka luo tietokannan ja asettaa sen aktiiviseksi.

`function GetCurrentDB: TFSDataBase;`

Palauttaa aktiivisen tietokannan.

`property galleryPath: String read gPath write gPath;`

Galleriahakemisto, johon haku oletuksena kohdistetaan.

7.11 DataBase.pas

Sisältää yleistä tietokantaa kuvaavat luokat.

`TDataBase = class`

Abstrakti tietokantaa kuvaava luokka, joka määrittelee hakumetodin muodon. Konkreettien tietokantaa kuvaavien luokkien on tarkoitus periä tämä.

`constructor Create; virtual; abstract;`

Virtuaalinen konstruktori.

```
procedure Find(addTo: TStringList; beg:string; filetypes:TStringList;
keywords:TStringList; dm:DoneMethod); virtual; abstact;
```

Virtuaalinen hakumetodi. Parametrit:

- `addTo`: merkkijonolista, johon saadut hakutulokset lisätään
- `beg`: etsittävän tiedoston nimen alku
- `filetypes`: lista etsittävistä tiedostotyypeistä
- `keywords`: lista avainsanoista, joita etsitään tiedostosta
- `dm`: metodi, jota kutsutaan kun haku on suoritettu

7.12 FSDataBase.pas

Sisältää luokat, jotka kuvaavat tietokantana käsiteltävää hakemistoa.

`TFindThread = class(TThread)`

Säie, jossa haku suoritetaan. Jokaiselle haulle luodaan oma säie, joka tuhotaan etsinnän päätyttyä.

`procedure AddStrings;`

Metodi, joka lisää (toisessa säikeessä olevaan) merkkijonolistaan saadut hakutulokset. Tätä metodia ajetaan synkronoituna.

`procedure Execute; override;`

Metodi, jolla säie käynnistetään ja joka suorittaa itse haun.

`procedure SetFindInfo(...);`

Asettaa säikeelle tarvittavat tiedot hakua varten.

`TFSDataBase = class(TDataBase)`

Tiedostojärjestelmää kuvaava tietokantaluokka. Perii luokan `TDataBase`. Suorittaa haun erillisessä säikeessä käyttäen `Libc`-kutsuja ja komentoja `find`, `grep` ja `head` sekä POSIX-shellin forsilmukkaa. Mikäli halutaan siirrettävyyttä Windows-ympäristöön, tulee tämä luokka korvata vastaavan haun suorittavalla luokalla, joka käyttää object pascalin tiedostonhallintakomentoja.

`constructor Create(dir:string); reintroduce;`

Konstruktori, joka ottaa parametrina tietokannan juurihakemiston.

`procedure Find(...);`

Luo hakusäikeen, asettaa säikeelle tarvittavat tiedot ja käynnistää sen.

`procedure SetRecursive(r: boolean);`

Asettaa haun rekursiiviseksi tai ei-rekursiiviseksi.

`procedure SetCurrentDir(d: string);`

Asettaa nykyisen hakemiston.

`function GetCurrentDir(): String;`

Palauttaa nykyisen hakemiston.

`function GetRoot: string;`

Palauttaa tietokannan juurihakemiston.

7.13 FileType.pas

Sisältää luokat, jotka kuvaavat tiedostotyyppejä. Tyyppien avulla asetetaan oletusarvoja projektiin liitettäville komponenteille. Verstas lukee käynnistyksen yhteydessä tiedostotyyppien tiedot `ftyperc`-tiedostosta.

`TMatchableString = class`

Apuluokka, jonka avulla merkkijonoja voidaan vertailla toisiinsa.

`function Match(st: string) : boolean;`

Palauttaa arvon tosi, jos parametrina annettu merkkijono on luokan kuvaaman merkkijonon osa.

`property str: string read fstr write fstr;`

Luokan kuvaama merkkijono tavallisena stringinä.

`TFileType = class`

Yhtä tiedostotyyppiä kuvaava luokka. Tiedostotyypillä on pattern-ominaisuus (esimerkiksi `.f90`) jonka perusteella tiedoston nimestä voidaan tutkia onko se tätä tyyppiä. Tiedostotyyppiin liittyvien ominaisuuksien, kuten `autoUpdate`-lipun, perusteella asetetaan projektiin liitettäville komponenteille oletusarvot, joita voi myöhemmin muuttaa komponenttikohtaisesti.

`TFileType` tietää myös merkkijonomuotoisen listan sovelluksista, joissa tämän tyyppisiä tiedostoja voidaan avata. Yksi sovelluksista on oletussovellus.

`function GetPattern: string;`

Palauttaa tiedostotyyppiä kuvaavan patternin. Käytetään `pattern`-ominaisuuden kautta.

`procedure SetPattern(const Value: string);`

Asettaa tiedostotyyppiä kuvaavan patternin. Käytetään `pattern`-ominaisuuden kautta.

`function GetDefaultApp: string;`

Palauttaa merkkijonona oletusalisovelluksen, jossa tämän tyyppiset tiedostot tulisi avata.

`procedure SetDefaultApp(app: string);`

Asettaa oletussovelluksen, jossa tämän tyyppiset tiedostot tulisi avata.

`function Match(st: string) : boolean;`

Palauttaa tiedon siitä, onko parametrina annettu tiedosto tätä tiedostotyyppiä.

`procedure AddApp(app: string);`

Lisää yhden sovelluksen, jossa tämän tyyppisiä tiedostoja voidaan avata.

`procedure GetApps(var list: TStringList);`

Palauttaa listassa kaikki sovellukset, joissa tämän tyyppiä tiedostoja voidaan avata.

`property description : string read fdescription write fdescription;`

Tiedostotyypin kuvaus merkkijonona.

`property defaultApp : string read GetDefaultApp write SetDefaultApp;`

Kts. `GetPattern` ja `SetPattern`.

`property defaultPhase : string read fdefaultPhase write fdefaultPhase;`

Oletusvaihe, johon tämän tyyppiset tiedostot oletetaan kuuluvaksi.

`property name: string read fname write fname;`

Tiedostotyypin nimi, jonka perusteella tyyppi tunnistetaan `ftyperc`-tiedostossa.

`property autoOpen : boolean read fautoOpen write fautoOpen;`

Lippu, joka kertoo tuleeko tämän tyyppiset tiedostot avata automaattisesti, kun ne liitetään projektiin.

`property autoUpdate : boolean read fautoUpdate write fautoUpdate;`

Lippu, joka kertoo tuleeko tämän tyyppisten tiedostojen muuttuneet tiedot välittömästi päivittää kaikissa sovelluksissa, joissa ne ovat auki.

7.14 FileTypeDialog.pas ja FileTypeDialog.xfm

`TFileTypeForm = class(TForm)`

Lomake, jolla valitaan etsittävät tiedostotyypit. Etsittävät tiedostotyypit valitaan tarkistusruuduilla (checkbox). Avataan kun etsintädialogista painetaan Select-painiketta. Aina kun tarkistusruutua klikataan, kutsutaan (etsintädialogin) metodia, joka on annettu konstruktorissa (parametrina tarkistusruudut). Näin etsintädialogi voi päivittää etsittävät tiedostotyypit.

`procedure CloseBClick(Sender: TObject);`

Suoritetaan, kun käyttäjä painaa Close-nappia.

`procedure FormResize(Sender: TObject);`

Kutsutaan kun lomakkeen kokoa muutetaan. Sijoittelee komponentit koon mukaan.

`procedure AddCheckBox(ft: TFileType; y: integer);`

Lisää tarkistusruudun lomakkeelle.

`constructor Create(owner: TComponent; m: BoxClickedMethod);`

Konstruktori jossa annetaan metodi jota kutsutaan tarkistusruutua painettaessa.

`procedure UpdateContents;`

Metodi, joka kysyy `AppManager`-luokalta tunnetut tiedostotyypit ja luo niille tarkistusruudut.

`procedure BoxClicked(sender: TObject);`

Metodi, jota kutsutaan kun tarkistusruutua klikataan.

`procedure SetBoxClicked(m: BoxClickedMethod);`

Asettaa metodin, jota kutsutaan kun tarkistusruutua klikataan.

7.15 Help.pas

HTML-formaatissa olevia avustustiedostoja näyttävä komponentti.

`TformHelp = class(TForm)`

`procedure LoadHelpFile(fileName: String);`

Avaa parametrina annetun avustustiedoston.

`procedure toolButtonBackClick(Sender: TObject);`

Siirtyy tiedostohistoriassa edelliseen dokumenttiin painettaessa Back-painiketta.

`procedure toolButtonFwdClick(Sender: TObject);`

Siirtyy tiedostohistoriassa seuraavaan dokumenttiin painettaessa Forward-painiketta.

`procedure toolButtonHomeClick(Sender: TObject);`

Siirtyy ensimmäiseen dokumenttiin.

`procedure copyMenuItemClick(Sender: TObject);`

Kopioi valitun tekstin leikepöydälle.

`procedure selectAllMenuItemClick(Sender: TObject);`

Valitsee koko dokumentin.

`procedure toolButtonExitClick(Sender: TObject);`

Sulkee avustustoiminnon.

7.16 Initializator.pas

Sisältää luokat Verstaan käynnistyksen yhteydessä tarvitsemien tietojen hallintaan. Tiedot luetaan asetustiedostosta käynnistyksen yhteydessä asetustiedostoista, jotka sijaitsevat joko käyttäjän kotihakemiston alla olevasssa `.verstas`-hakemistossa, tai mikäli hakemistoa ei vielä ole luotu, samassa hakemistossa Verstas-sovelluksen kanssa.

`TInitializator = class`

Singleton-tyyppinen luokka, joka hoitaa mm. asetustustiedostojen lukemisen ja tallentamisen sekä Verstaan sulkeutuessa muiden singleton-tyyppisten luokkien tuhoamisen.

`procedure Initialize;`

Lukee asetustiedostot ja luo Verstaan sisälle vastaavat oliot, kuten `AppManger`in sisältämät `TAppHandle`t ja `TFileType`t.

`procedure FreeAll;`

Kutsuu kaikkien muiden singleton-tyyppisten luokkien destruktoreja, jotka puolestaan vapauttavat luomansa oliot.

`procedure WriteIni;`

Kirjoittaa asetustiedostot muutuneiden tietojen mukaisesti.

7.17 ItemSettingsDialog.pas ja ItemSettingsDialog.xfm

Lomake ja koodi dialogille, jolla voidaan katsella ja asettaa projektisolmun tietoja.

`TformItemSettings = class(TForm)`

`procedure EditItem(item: TProjectItem);`

Asetusdialogia käytetään kutsumalla tätä metodia ja antamalla syötteeksi viite muokattavaan projektisolmuun. Dialogi aukeaa tällöin modaalisena ja solmun tiedot muuttuvat, mikäli käyttäjä poistuu painamalla hyväksymispainiketta.

7.18 MainFrame.pas ja MainFrame.xfm

Sovelluksen graafinen käyttöliittymä.

`TformMain = class(TForm)`

TformMain on sovelluksen päälomaketta vastaava luokka. Seuraavassa on esitelty luokan ulkopuolelta kutsuttavissa olevat metodit.

`procedure ClearServices;`

Tyhjentää valikon, joka sisältää kaikkien käynnistettävissä olevien alisovellusten tarjoamat palvelut.

`procedure AddService(definition: string; tag: integer);`

Lisää alisovelluksen tarjoaman palvelun valikoon. Parametreina palvelun kuvaus sekä kokonaislukutunniste, jonka perusteella palvelua vastaava sovellus voidaan aktivoida.

`procedure AddWnd(cap: string; tag: integer);`

Lisää ikkunan Windows-valikkoon. Parametreina valikkoon tuleva ikkunan nimi sekä ikkunaa vastaava kokonaislukutunniste, jonko perusteella ikkuna voidaan aktivoida.

`procedure RemoveWnd(tag: integer);`

Poistaa ikkunan Windows-valikosta kokonaislukutunnisteen perusteella.

`procedure ShowInfo(txt: string);`

Näyttää käyttäjälle parametrina annetun merkkijonon.

7.19 Project.pas

Luokka, joka kapseloi yhden mallinnusprojektin tiedot.

`TProject = class(TObject)`

`procedure NotifyContentsChanged;`

Ilmoittaa Verstaan sisäisellä viestillä, että projektin sisältö on muuttunut.

`procedure NotifyPhaseChanged;`

Ilmoittaa Verstaan sisäisellä viestillä, että projektista on valittu aktiiviseksi uusi vaihe.

`procedure ReadFromFile(filename: string);`

Lukee tiedostoon tallennetun projektin.

`procedure Save;`

Tallentaa projektin samalle nimelle.

`procedure SaveAs(projectfilename: string);`

Tallentaa projektin syötteenä annettavalle tiedostonimelle.

`procedure Attach(Comp : TProjectComponent); overload;`

Liittää projektiin komponentin, joka on luotu jossakin toisaalla.

`procedure Attach(Comp : TProjectComponent; Phase: TProjectPhase);overload;`

Liittää projektiin komponentin, joka on luotu jossakin toisaalla. Komponentti liitetään suoraan syötteenä annettavaan vaiheeseen.

`procedure Detach(Comp : TProjectComponent);`

Irrottaa komponentin projektista. Komponenttia ei vapauteta, vaan se tulee tehdä muualla.

`procedure FreeComponent(Comp : TProjectComponent);`

Irrottaa komponentin projektista ja vapauttaa komponentin.

`function ComponentExists(comp: TProjectComponent) : boolean;`

Palauttaa tosi, jos syötteenä annettu komponentti kuuluu tähän projektiin.

`procedure GetComponentsByFilenames(var sl: TStringList);`

Kutsun jälkeen syötteenä annettua merkkijonolistaa voidaan käyttää merkkijono-arvo -parien listana. Se sisältää kuvauksen tiedostonimiltä komponentteja vastaaville olioille.

```
function NewPhase(phasename: string; phaseparent: TProjectPhase) :
TProjectPhase; overload;
```

Projektiin liitetään uusi alivaihe.

`function NewPhase(phasename: string): TProjectPhase; overload;`

Projektiin liitetään uusi päävaihe.

`function NewPhase(phase : TProjectPhase) : TProjectPhase; overload;`

Projektiin liitetään uusi päävaihe, joka on luotu toisaalla.

`function PhaseExists(phase: TProjectPhase) : boolean;`

Palauttaa tosi, jos syötteenä annettu vaihe kuuluu tähän projektiin.

`function GetPhaseByName(phasename: string) : TProjectPhase;`

Palauttaa vaihesolmun, jonka nimi on vastaa syötteenä annettua.

`procedure CopyMainPhaseList(var mainphaselist : TList);`

Kutsun jälkeen syötteenä annettu lista sisältää viittaukset projektin päävaiheisiin.

`function GetPhaseNumber(phase: TProjectPhase) : integer;`

Funktio palauttaa indeksin siihen päävaiheeseen, johon syötteenä annettu vaihe kuuluu.

`procedure GetPhaseNames(var names: TStringList);`

Kutsun jälkeen syötteenä annettu merkkijonolista sisältää projektin kaikkien vaiheiden nimet.

`procedure SetAssociation(prjcomp:TProjectComponent; phase: TProjectPhase);`

Liittää komponentin vaiheeseen. Käytännössä siis asettaa syötekomponentin vanhemmaksi syötevaiheen.

`function GetPhaseOfComponent(comp: TProjectComponent) : TProjectPhase;`

Palauttaa vaiheen johon komponentti kuuluu tai `nil`, jos komponentin vanhempi ei ole vaihe.

`function HasItem(item: TProjectContainerItem) : boolean;`

Palauttaa tosi, jos syötteenä annettu solmu kuuluu projektiin.

`property name: string read Getname write Setname;`

Projektin nimi.

`property description: string read Getdescription write Setdescription;`

Projektin kuvaus.

`property filename: string read Getfilename write Setfilename;`

Projektin kuvaustiedoston tiedostonimi.

`property workdirectory: string read Getworkdirectory;`

Projektihakemisto, joka sisältää kaikki komponentit.

`property initialized: boolean read Getinitialized;`

Palauttaa tosi, jos projekti on kunnolla luotu tai ladattu, eli sitä voidaan käyttää.

`property isModified: boolean read FisModified;`

Lippu, joka kertoo onko projektin rakenne muuttunut edellisen tallennuksen jälkeen tai onko tallennuksen jälkeen muokattu komponentteja.

`property MainPhaseCount: Integer read GetMainPhaseCount;`

Projektin päävaiheiden lukumäärä

`property MainPhases[i: Integer]: TProjectPhase read GetMainPhase;`

Indeksoitu päävaiheiden saanti.

`property Root: TProjectRoot read Froot;`

Juurisolmun suora saanti.

`property ActivePhase: TProjectPhase read GetactivePhase write SetactivePhase;`

Tällä hetkellä aktiivinen vaihe.

`property ActiveMainPhase: TProjectPhase read GetactivemainPhase;`

Päävaihe, joka sisältää aktiivisen vaiheen.

7.20 ProjectComponent.pas

Mallinnuskomponentin kapselointi, joka on peritty yleisestä projektisolmusta (*project item*)

`TProjectComponent = class(TProjectItem)`

`procedure DoAssignFromStringList(SL: TStringList); override;`

Kutsuu perittyä asetusmetodia ja lisää omien kenttien asetuksen.

`function GetAsString: string; override;`

Kutsuu perittyä tulostusmetodia ja lisää omien kenttien tulostuksen.

`property filename: string read Getfilename write Setfilename;`

Tätä mallinnuskomponenttia vastaavan tiedoston nimi.

`property path: string read GetPath;`

Ominaisuudesta voi lukea tiedoston sijaintipolun.

`property application: string read Fapplication write Setapplication;`

Mihin alisovellukseen tämä komponentti avataan aktivoitaessa.

`property isModified: boolean read FisModified write SetisModified;`

Tätä lippua voi käyttää tallentamattomien tietojen kirjanpitoon. Käytännössä tällä hetkellä ominaisuus ei ole luotettava, vaan tulee kysyä alisovelluksista muuttuneita tietoja.

`property topPhase: TProjectContainerItem read gettopPhase;`

Palauttaa päävaiheen (ylimmän tason vaiheen) johon tämä komponentti kuuluu. Palauttaa nil, jos komponentti ei kuuluu mihinkään vaiheeseen.

`property openIn: TAppHandle read getAppHandle write setAppHandle;`

Ominaisuus vastaa sen alisovelluksen kahvaa, johon tämä komponentti avataan aktivoitaessa.

`property autoUpdate : boolean read fautoUpdate write fautoUpdate;`

Ominaisuus kertoo päivitetäänkö komponentin tiedot kaikissa alisovelluksissa automaattisesti, jos jossakin niistä tallennetaan uusi versio komponentista.

7.21 ProjectItems.pas

Käännösyksikkö `ProjectItems` sisältää kantaluokat kaikkien projektiin liitettävien asioiden, kuten vaiheet ja komponentit, tallentamiseksi. Tiedot tallennetaan solmuihin (*item*) joilla voi olla lapsia ja yksi vanhempi (paitsi juurisolmulla ei ole vanhempaa eikä mallinnuskomponentilla lapsia). Käännösyksikkö sisältää kantaluokkien lisäksi konkreettisen luokan projektin juurisolmulle.

`TProjectItem = class(TObject)`

TProjectItem on kantaluokka kaikille projektin tietojen tallennukseen tarkoitetuille luokille.

`procedure Setparent(const Value: TProjectContainerItem); virtual;`

`Parent`-ominaisuuden asetusmetodi, joka päivittää edellisen ja uuden vanhemman tiedot.

`property parent : TProjectContainerItem read Fparent write Setparent;`

Tämän solmun vanhempi

`property name : String read Fname write Setname;`

Projektisolmulla on ominaisuutena nimi, joka voidaan näyttää käyttäjälle.

`property description : String read Fdescription write Setdescription;`

Projektisolmulla on ominaisuutena pidempi ihmisen luettava kuvaus, joka voidaan näyttää käyttäjälle.

`property tagNumber : Integer read FtagNumber write SettagNumber;`

Numero, jota voi periaatteessa käyttää mihin vain.

`property topContainer : TProjectContainerItem read Gettopcontainer;`

Ominaisuudesta voi lukea mikä on solmun kaukaisin esivanhempi puurakenteessa.

`property topRoot : TProjectRoot read GettopRoot;`

Jos kaukaisin esivanhempi on juurisolmu, juuren saa tästä ominaisuudesta. Muutoin ominaisuus on `nil`.

`procedure DoAssignFromStringList(SL : TStringList); virtual;`

Metodin `AssignFromString` käyttämä metodi: kenttien luku merkkijonolistasta. Perityt luokat kirjoittavattästä aliohjelmasta oman versionsa, joka kutsuu aluksi perittyä funktiota.

`procedure AssignFromString(input: string); virtual;`

Asettaa olion kentät syötteenä annettavasta merkkijonosta käyttäen metodia `DoAssignFromString`. Merkkijonon formaatti on sellainen kuin saadaan ulos käyttämällä `GetAsString`-metodia.

`function GetAsString: string; virtual;`

Kenttien tulostus merkkijonoon. Informaatio solmujen lapsista ja vanhemmista ei tulostu muiden kenttien mukana vaan se täytyy hakea erikseen.

`TProjectContainerItem = class(TProjectItem)`

Kantaluokka sellaisille solmuille, joilla on lapsia.

`procedure AddChild(item : TProjectItem); virtual;`

Sisäinen metodi, jota ei tule kutsua suoraan.

`procedure RemoveChild(item : TProjectItem); virtual;`

Sisäinen metodi, jota ei tule kutsua suoraan.

`property childCount : Integer read GetchildCount;`

Ominaisuudesta voi lukea tämän solmun lasten määrän.

`property children[i:integer] : TProjectItem read Getchildren;`

Lasten indeksoitu saanti.

`property compCount: Integer read GetCompCount;`

Ominaisuudesta voi lukea tämän solmun lapsina olevien mallinnuskomponenttien määrän.

`property comps[i: Integer]: TProjectItem read GetComps;`

Lapsikomponenttien indeksoitu saanti.

`property phaseCount: Integer read GetPhaseCount;`

Ominaisuudesta voi lukea tämän solmun lapsina olevien vaiheiden määrän.

`property phases[i: Integer]: TProjectItem read GetPhases;`

Lapsivaiheiden indeksoitu saanti

`function IndexOfPhase(phase : TProjectItem) : Integer;`

Palauttaa syötteenä annetun vaiheen indeksin.

`TProjectRoot = class(TProjectContainerItem)`

Projektin juurisolmulla ei voi olla vanhempaa. Sen nimi ja kuvaus vastaavat koko projektin nimeä ja kuvausta. Juurisolmu vastaa projektia käyttäjän näkymässä.

`procedure DoAssignFromStringList(SL: TStringList) ; override;`

Juurisolmussa on uudelleenkirjoitettu `DoAssignFromStringList`, jolla sen omat kentät saadaan asetettua.

`function GetAsString: string; override;`

Juurisolmussa on uudelleenkirjoitettu `GetAsString`, jolla sen omat kentät saadaan tulostettua.

`property workdir : string read Fworkdir write Setworkdir;`

Projektin työhakemisto.

`property filename : string read Ffilename write Setfilename;`

Projektitiedoston nimi.

7.22 ProjectManager.pas

Käännösyksikkö `ProjectManager` sisältää välineet useiden mallinnusprojektien yhdenaikaiseen hallintaan.

`TProjectManager = class(TObject)`

Manageria tulee käyttää yksilöllisen instanssin (`TProjectManager.Instance`) kautta.

`class function Instance: TProjectManager;`

Luokkafunktio, joka palauttaa osoittimen manageriolioon. Funktion ensimmäinen kutsu tekee uuden manageriolion.

`procedure AddProject(prj : TProject);`

Sisäinen metodi: Syötteenä annettava projekti liitetään managerissa avoinna olevien projektien listaan, mutta listasta ennestään löytyvää projektia ei lisätä uudestaan.

`procedure RemoveProject(prj : TProject);`

Sisäinen metodi: Syötteenä annettava projekti poistetaan managerissa avoinna olevien projektien listasta. Tämän kutsuminen ei tuhoa projektia, vaan se on tehtävä muualla.

`procedure NotifyStateChanged;`

Tämä metodi lähettää Verstaan sisäisen `TVCmdNotifyProjectState`-ilmoituksen.

`function OpenProject(fileName: string) : TProject;`

Funktio avaa projektitiedoston, johon syötteenä annettava tiedostonimi viittaa. Tiedostonimen tulee sisältää polku. Jos projekti oli mahdollista avata, se liitetään managerin listaan ja viittaus uuteen projektiin annetaan palautusarvona.

`procedure SaveProject;`

Tallentaa aktiivisen projektin samalle nimelle kuin ennenkin.

`procedure SaveProjectAs(fileName: string);`

Tallentaa aktiivisen projektin nimelle, joka annetaan syötteenä.

`function NewProject(directory: string; name: string) : TProject;`

Luo ja alustaa uuden projektin sekä lisää sen projektilistaan. Palauttaa viittauksen uuteen projektiin.

`procedure CloseProject(project: TProject);`

Sulkee syötteenä annettavan projektin.

`procedure GetPhaseNames(var names: TStringList);`

Metodikutsun jälkeen syötteenä annettavassa merkkijonolistassa on kaikkien avoinna olevien projektien kaikki vaiheet.

`function GetProjectByRoot(root: TProjectRoot) : TProject;`

Hakee projektilistasta sen projektiolion, jonka juurisolmu on annettu syötteenä.

`function GetComponentByFileName(filename : string) : TProjectComponent;`

Hakee mallinnuskomponentin, jonka nimi hakemistopollkuineen vastaa syötteenä annettua.

`procedure GetAllProjects(var list: TList);`

Kutsun jälkeen syötteenä annettu listan sisältö on sama kuin avoinna olevien projektien lista.

`property projectCount: integer read GetProjectCount;`

Avoinna olevien projektien määrä eli projektilistan koko.

`property projects[i: integer]: TProject read GetProjects ;`

Projektiolioiden saanti projektilistasta indeksin avulla.

`property activeProject: TProject read factiveProject write SetActiveProject;`

Tällä hetkellä aktiivisena olevan projektin saanti. Ominaisuus on `nil`, mikäli mikään projekti ei ole aktiivinen.

`property activeProjectRoot: TProjectRoot read getactiveProjectRoot;`

Aktiivisen projektin juurisolmun suora saanti. Ominaisuus on `nil`, mikäli mikään projekti ei ole aktiivinen.

`property projectsModified: boolean read GetProjectsModified;`

Ominaisuus on tosi, mikäli jokin avatuista projekteista sisältää tallentamatonta tietoa.

`property maxProjects: integer read FMaxProjects write SetMaxProjects;`

Yhtä aikaa avoinna olevien projektien maksimimäärä.

`property DefaultVprFile: string read FDefaultVprFile write SetDefaultVprFile;`

Oletusprojektitiedoston nimi. Oletusprojektitiedosto luetaan uuden projektin lähtökohdaksi.

7.23 ProjectPhase.pas

Käännösyksikkö `ProjectPhase` sisältää luokan, joka kapseloi projektin vaiheen ominaisuudet.

`TProjectPhase = class(TProjectContainerItem)`

`TProjectPhase` on peritty `TProjectContainerItem`istä, joten se on projektisolmu, jolla voi olla lapsia.

`function Gettopphase: TProjectPhase;`

Sisäinen saantimetodi `topphase`-ominaisuudelle.

`property parent;`

Vaiheella voi olla vanhempi, joten tämä `TProjectItem`issä toteutettu ominaisuus tehdään sille julkiseksi.

`property topphase : TProjectPhase read Gettopphase;`

`Topphase`-ominaisuudesta voi lukea mihin päävaiheista tämä (pää- tai alivaihe) kuuluu.

7.24 ProjectView.pas

Käännösyksikkö `ProjectView` sisältää projektin vaiheita ilmentävän välilehtinäkymän (*tab control*).

`type TProjectPhaseView = class(TTabControl)`

Mallinnusprojektin vaiheita ilmentävä käyttöliittymäkomponentti on peritty Kylixin omasta välilehtinäkymästä (`TTabControl`) ja siten olennaisesti eri tavalla toteutettu kuin puunäkymäkehys.

`procedure VCmdNotifyProjectContentsHandler(cmd: TCmd);`

Verstaan sisäisen ilmoituksen käsittelijä, jonka tehtävä on päivittää näkymä tarvittaessa.

`procedure VCmdNotifyProjectStateHandler(cmd: TCmd);`

Verstaan sisäisen ilmoituksen käsittelijä, jonka tehtävä on päivittää näkymä tarvittaessa.

`procedure Changed(Value: Integer); override;`

`TTabControl`'issa olevan metodin uusi määrittely, jota kutsutaan kun käyttäjä vaihtaa välilehteä. Metodi suorittaa Verstaan sisäisen komennon, jolla projektin aktiivinen vaihe vaihdetaan.

`procedure Refresh;`

Varsinainen päivitys vastaamaan projektimanagerin nykyistä tilaa.

7.25 ProjectViewFrame.pas

`TFrameProjectComponentView = class(TFrame)`

`TFrameProjectComponentView` on peritty Kylix/Delphi - työkalun tarjoamasta kehys-luokasta (`TFrame`). Kehykset näkyvät Kylix-ohjelmakehitysympäristössä ikään kuin omana lomakkeenaan. Niihin voidaan sisällyttää komponentteja, joille voi myös tehdä tapahtumankäsittelijöitä. Kehykset voidaan liittää osaksi tavallista lomaketta, jolloin niillä on omat alkuperäiset ominaisuutensa ja lisäksi lomakkeella määriteltävät lisäominaisuudet.

Tässä tapauksessa kehykselle on sijoitettu puunäkymä ja ponnahdusvalikko (*popup-menu*). Tapahtumankäsittelijöiden avulla on mukautettu klikkaus- ja raahausoperaatioita sekä näkymän piirtoa kuvaruudulle. Ponnahdusvalikko sidotaan kontekstiin tapahtumakäsittelijän avulla.

Kattava tietous tapahtumankäsittelijöistä, niiden syötteistä ja kutsutilanteista on löydettävissä Kylixin ohjejärjestelmästä sekä Delphi-oppaista.

```
procedure TreeView1DragOver(Sender, Source: TObject; X, Y: Integer; State:
TDragState; var Accept: Boolean);
```

Tapahtumankäsittelijä, johon suoritus tulee käyttäjän raahatessa jotakin puunäkymän yllä. Tapahtumankäsittelijä ratkaisee hiiren sijainnin (X,Y) mukaan, sallitaanko raahaus vai ei.

`procedure TreeView1ContextPopup(Sender: TObject; MousePos: TPoint;var Handled: Boolean);`

Tapahtumankäsittelijä, joka suoritetaan ennen ponnahdusvalikon näyttämistä käyttäjälle. Täällä mukautetaan ponnahdusvalikon sisältö asiayhteyttä vastaavaksi.

`procedure TreeView1StartDrag(Sender: TObject; var DragObject: TDragObject);`

Tapahtumankäsittelijä, joka suoritetaan käyttäjän aloittaessa raahaustoiminnon.

`procedure TreeView1DragDrop(Sender, Source: TObject; X, Y: Integer);`

Tapahtumankäsittelijä, joka suoritetaan käyttäjän lopettaessa raahaustoiminnon hyväksyttävään pudotukseen. Täällä määritellään Verstaan klikkaus ja raahaus -toiminnallisuus.

`procedure TreeView1CustomDrawItem(Sender: TCustomViewControl;Item: TCustomViewItem; Canvas: TCanvas; const Rect: TRect; State: TCustomDrawState; Stage: TCustomDrawStage; var DefaultDraw: Boolean);`

Tapahtumankäsittelijä, jolla mukautetaan puunäkymän piirto kulloinkin piirrettävän solmun sisällön mukaan.

`procedure TreeView1ItemEnter(Sender: TObject; Node: TTreeNode);`

Tämä tapahtumankäsittelijä herättää vinkkinäkymän (*hint window*) jossa kerrotaan hiiren alle tulleen solmun kuvaus.

`procedure TreeView1ItemExitViewportEnter(Sender: TObject);`

Tapahtumankäsittelijä piilottaa vinkkinäkymän.

`procedure CopyComponentToActiveProject(Sender: TObject);`

Ponnahdusvalikon valinnan `CopyToActiveProject` tapahtumankäsittelijä, joka luo Verstaan sisäisen komennonn valitun komponentin kopioimiseksi aktiiviseen projektiin

`procedure VCmdNotifyProjectContentsHandler(cmd: TCmd);`

Käsittelijä Verstaan sisäiselle ilmoitukselle projektin sisällön muuttumisesta. Päivittää näkymän.

`procedure VCmdNotifyProjectStateHandler(cmd: TCmd);`

Käsittelijä Verstaan sisäiselle ilmoitukselle projektin tilan muuttumisesta. Päivittää näkymän.

`constructor Create(AOwner: TComponent); override;`

Normaali konstruktori, jossa varataan resurssit ja asetetaan sopivat alkuarvot sekä ilmoittaudutaan kuuntelemaan Verstaan sisäisiä ilmoituksia.

`procedure Rebuild;`

Metodi, joka rakentaa koko puunäkymän sisällön uudestaan projektimanagerin sisällön mukaiseksi.

`procedure Refresh;`

Metodi, jota kutsutaan kun puunäkymä tai osa siitä halutaan päivittää. Tämän hetkinen käytännön toteutus vain suorittaa uudelleenrakennuksen (`Rebuild`).

7.26 PWnd.pas

Sisältää kantaluokan sellaisille ikkunoille, jotka liittyvät johonkin projektiin.

`TPWnd = class (TWnd)`

Abstrakti luokka kuvaamaan sellaista ikkunaa, joka liittyy tiettyyn projektiin. Tällöin ikkuna myös mahdollisesti liittyy johonkin projektin vaiheeseen, ja tulee näyttää vain silloin, kun kyseinen vaihe on aktiivinen. Luokka sisältää metodin, joka muuttaa projektin vaiheen työpöydän numeroa kuvaavaksi kokonaisluvuksi.

`function GetParent : TProjectContainerItem; virtual; abstract;`

Palauttaa ikkunan parentin, joka on jokin projektin osa, yleensä jokin projektin vaihe tai sen juuri.

`function GetDesktop: integer; override;`

Palauttaa kokonaislukuna työpöydän, jolle ikkuna kuuluu. Metodi kutsuu virtuaalista `GetParent`-metodia, jonka konkreettiset ikkunaluokat toteuttavat. Saamansa `parent`-olion avulla metodi selvittaa `ProjectManager`ilta, mille välilehdelle ikkuna kuuluu.

`function GetShowInAll: boolean; override;`

Palauttaa tiedon siitä, tuleeko ikkuna näyttää kaikilla työpöydillä.

`property parent: TProjectContainerItem read GetParent;`

Ominaisuus ikkunan parentin selvittämiseksi. Kutsuu `GetParent`-metodia.

7.27 SearchDialog.pas ja SearchDialog.xfm

`TSearchForm = class(TForm)`

Etsinnän suorittamiseen käytettävä lomake. Sisältää etsinnän suorittamiseen ja tulosten näyttämiseen tarvittavat käyttöliittymäkomponentit. Ohjaa itse haun `TDBManager`-luokan sisältämälle tietokannalle (`TFSDatabase`). Käyttää etsittävien tiedostotyyppien valitsemiseen `TFileTypeForm`-luokkaa.

`procedure CloseButtonClick(Sender: TObject);`

Metodi, joka suoritetaan kun Close-nappia painetaan. Sulkee etsintädialogin.

`procedure SelectRadioBClick(Sender: TObject);`

Suoritetaan kun käyttäjä painaa Select-nappia halutessaan valita etsimänsä tiedostotyypit. Avaa tiedostotyyppidialogin (`TFiletypeForm`).

`procedure FormCreate(Sender: TObject);`

Suoritetaan kun lomake luodaan.

`procedure AllRadioBClick(Sender: TObject);`

Suoritetaan kun käyttäjä klikkaa All-nappia (valitessaan etsivänsä kaikkia tiedostotyyppejä).

`procedure DirTreeChange(Sender: TObject; Node: TTreeNode);`

Metodi, joka suoritetaan kun käyttäjä vaihtaa hakemistoa puunäkymässä.

`procedure RecursiveCheckBClick(Sender: TObject);`

Suoritetaan kun käyttäjä klikkaa Recursive-nappia.

`procedure SearchEditKeyUp(Sender: TObject; var Key: Word;`

Suoritetaan kun käyttäjä kirjoittaa (tai poistaa) kirjaimen etsittävän tiedoston nimeen. Kutsuu hakutulosten päivitystä (`UpdateResults`).

`procedure AddButtonClick(Sender: TObject);`

Metodi, joka suoritetaan kun käyttäjä klikkaa Add to Project -nappia. Luo tiedoston lisäämiseen käytettävän komennon jokaiselle valitulle tiedostonimelle ja suorittaa sen.

`procedure FormDestroy(Sender: TObject);`

Kutsutaan kun lomake tuhotaan.

`procedure keyWordEditKeyUp(Sender: TObject; var Key: Word;`

Metodi jota kutsutaan, kun käyttäjä kirjoittaa kirjaimen tiedostosta etsittävään hakusanaan.

`procedure FormResize(Sender: TObject);`

Kutsutaan kun lomakkeen kokoa muutetaan. Kutsuu `ArrangeComponents`ia, jotta komponentit järjestetään uusiksi.

`procedure FormClose(Sender: TObject; var Action: TCloseAction);`

Kutsutaan kun lomake suljetaan.

`procedure UpdateResults;`

Metodi, joka pyytää tietokantaa (`TFSDatabase` suorittamaan haun valituilla parametreillä ja päivittää tulokset niille varattuun tilaan.

`procedure UpdateKeyWords;`

Päivittää haulle annettavat hakusanaparametrit.

`procedure SetDone;`

Ilmoittaa käyttäjälle, että haku on suoritettu.

`procedure SetSearching;`

Ilmoittaa käyttäjälle, että hakua suoritetaan.

`procedure ArrangeComponents;`

Järjestelee lomakkeen komponentit uudelleen lomakkeen koon mukaan.

`procedure UpdateFileTypes(cboxes: Tlist);`

Päivittää haulle annettavat etsittävät tiedostotyypit.

7.28 SettingsDialog.pas ja SettingsDialog.xfm

Sisältää dialogin ja vastaavan luokan käyttäjäkohtaisten asetusten muuttamiseen ja tallentamiseen.

`TformSettings = class(TForm)`

Asetusten muuttamisesta vastaava luokka.

`procedure FormCreate(Sender: TObject);`

Lomakkeen luonnin yhteydessä kutsuttava `FormCreate` luo asetuslomakkeelle dynaamisesti kontrollit tiedostotyyppien oletusasetusten muuttamiseksi.

`procedure buttonOKClick(Sender: TObject);`

OK-napin painalluksen käsittelijä, jossa käyttäjän säätämät asetukset siirretään asianomaisiin olioihin, joista ne Verstaasta poistuttaessa tallennetaan `ftyperc`-tiedostoon.

`procedure buttonCancelClick(Sender: TObject);`

Cancel-napin painalluksen käsittelijä, joka sulkee lomakkeen päivittämättä muuttuneita tietoja.

`procedure FormDestroy(Sender: TObject);`

Vapauttaa varatut tietorakenteet.

7.29 Utils.pas

Sisältää yleiskäyttöisiä apuohjelmia.

`function GetNextQuotation (var param: string) : string;`

Palauttaa seuraavan "-merkein erotetun osan parametrina saamastaan merkkijonosta. Poistaa merkkijonosta kaiken toiseen "-merkkiin saakka.

`function JoinDir(part1: string; part2: string): string;`

Liittää kaksi merkkijonoa toisiinsa siten, että niiden väliin jää tasan yksi hakemistoerotin. Erotin riippuu laitteistosta, jolla sovellus käännetään, ja on siis joko / tai \.

`function FileCopy(const FileFrom, FileTo: string) : boolean;`

Kopioi ensimmäisenä parametrina annetun tiedoston toisena paremetrina annetulle nimelle. Palauttaa tosi mikäli kopiointi onnistuu.

`procedure ShowError(msg: String);`

Näyttää annetun virheilmoituksen erillisessä dialogissa.

`function ExplodeTwo(input:string; var A, B:string; sepstr : string) : boolean;`

Jakaa parametrina annetun merkkijonon kahteen osaan annetun erottimen kohdalta. Erotinta ei jätetä kumpaankaan merkkijonoon. Palauttaa tosi, mikäli operaatio onnistuu.

7.30 VCmd.pas

Sisältää luokkina Verstaan sisäiset komennot. `VCmd`:sta perittyjen luokkien avulla on pyritty toteuttamaan kaikki sellaiset komennot, joiden suorittaminen tai komentoon liittyvät parametrit voivat kiinnostaa myös muita kuin kutsuja- ja suorittajaosapuolia.

`TNewComponent = class`

Apuluokka, jota käytetään uusien komponenttien tilapäiseen säilytykseen `TVCmdProjectAddFiles`-luokassa.

7.30.1 Abstaktit kantaluokat

`TVCmd = class(TCmd)`

Kaikkien Verstas-komentojen kantaluokka.

`TVCmdProject = class(TVCmd)`

Projektien hallintaan liittyvien komentojen kantaluokka.

`TVCmdApp = class(TVCmd)`

Alisovellusten hallintaan liittyvien komentojen kantaluokka.

`TVCmdUpdate = class (TVCmd)`

Päivityspyyntöinä käytettävien komentojen kantaluokka.

`TVCmdNotify = class (TVCmd)`

Ilmoituksina käytettävien komentojen kantaluokka.

`TVCmdWnds = class (TVCmd)`

Ikkunoiden hallintaan liittyvien komentojen kantaluokka.

`TVCmdFile = class (TVCmd)`

Tiedostojen hallintaan liittyvien komentojen kantaluokka.

`constructor Create(a:TApp; filename:string);`

Konstruktorin parametrina annetaan sovellus, joka tiedostoa käsittelee, sekä tiedoston nimi.

7.30.2 Konkreettiset komentoluokat

Kaikissa komentoluokissa on abstaktin `TCmd`-luokan abstraktin `DoExecute`-metodin korvaava metodi, jota kutsutaan, kun komento suoritetaan.

`TVCmdExit = class (TVCmd)`

Komento, joka suoritetaan kun Verstasta ollaan sulkemassa. Sulkee projektit ja huolehtii, että muutokset talletetaan käyttäjän niin halutessa, sekä käskee `Initilizatoria` kirjoittamaan käyttäjän asetukset tiedostoihin.

`TVCmdAppLaunch = class (TVCmdApp)`

Sovelluksen käynnistämiseen käytettävä komento. Komentoa kutsutaan, kun käyttäjä valitsee valikosta jonkin uuden komponentin luomisen.

`constructor Create(tag: integer);`

Konstruktorin parametrina annetaan käynnistettävän sovelluksen kokonaislukutunniste.

`TVCmdAppCloseProjectApps = class (TVCmdApp)`

Komento, jonka avulla suljetaan tiettyyn projektiin liittyvät sovellukset.

`constructor Create(project : TProject);`

Konstuktorin parametrina annetaan projekti, johon kuuluvat sovellukset tulee sulkea.

`property error: boolean read ferror;`

Ominaisuus asetetaan todeksi, komennon suorittamisen jälkeen, mikäli kaikkia asiaankuuluvia sovelluksia ei saatu suljettua.

`property cancelled : boolean read fcancelled;`

Ominaisuus asetetaan todeksi komennon suorittamisen jälkeen, mikäli käyttäjä perui toiminnon.

`TVCmdAppCreateInstance = class (TVCmdApp)`

Komento luodaan ja suoritetaan, kun jokin sovellus on ilmoittanut Verstaalle käynnistyneensä. Pyytää `AppManagerilta` sovellusta vastaavan olion, aktivoi sovellusta vastaavan projektin vaiheen sekä luo sovellusta vastaavan `TWnd`-olion, mikäli sovellus toteuttaa `gui`-rajapinnan.

`constructor Create(threadId: Cardinal; params: array of string);`

Konstruktorin parametreina annetaan käynnistyneen sovelluksen säietunniste, sekä ilmoituksen mukana saapuneet parametrit, eli sovelluksen nimi ja rajapinnat.

`TVCmdAppDeleteInstance = class (TVCmdApp)`

Poistaa Verstaasta sovellusta vastaavan olion sekä vastaavan `TWnd`-olion, mikäli sellainen on.

`constructor Create(threadId: Cardinal);`

Konstruktorin parametrina annetaan poistettavan sovelluksen säietunniste.

`TVCmdAppDeleteAllInstances = class (TVCmdApp)`

Poistaa kaikkien Verstaasta kaikkia sovelluksia kuvaavat oliot ja vastaavat ikkunaoliot.

`TVCmdProjectNew = class(TVCmdProject)`

Komento uuden projektin luomiseksi. Luo projektille uuden hakemiston, sekä käskee `ProjectManageria` luomaan projekti-olion.

`constructor Create(directory: string; name: string);`

Konstruktorin parametrina annetaan hakemistopolku, sekä luotavan alihakemiston nimi.

`TVCmdProjectOpen = class(TVCmdProject)`

Komento tallennetun projektin avaaamiseksi.

`constructor Create(fileName: string);`

Konstuktorin parametrina annetaan avattavan projektitiedoston nimi.

`TVCmdProjectClose = class(TVCmdProject)`

Komento projektin sulkemiseksi.

`constructor Create(projectrootitem: TProjectRoot);`

Parametrina annetaan suljettavan projektin juurisolmu.

`property project : TProject read fproject;`

Ominaisuuden avulla voidaan saada selville suljettava projekti.

`TVCmdProjectAddFiles = class(TVCmdProject)`

Komento tiedostojen lisäämiseksi projektiin. Kometo luo tiedostoja vastaava komponentit, mikäli tiedostot eivät jo kuulu projektiin, ja kopioi tiedostot projektihakemistoon.

`function CreateNewComponent(fileName: string): TNewComponent;`

Luo uuden väliaikaisen komponentti-olion. Olioon talletetaan luodun komponentin lisäksi tieto siitä, tuleeko komponentti aktivoida välittömästi projektiin liitettäessä, sekä projektin solmu, jonka alaisuuteen komponentti todennäköisesti tullaan liittämään.

`constructor Create(fileName: string; move: boolean = false); overload;`

Konstuktori, jonka avulla voidaan projektiin liittää yksittäinen annettu tiedosto. `Move`-parametri kertoo tuleeko komponentti kopioida projektihakemistoon.

`constructor Create(files: TStringList; move: boolean = false); overload;`

Konstuktori, jonka avulla projektiin voidaan liittää useita `TStringList`:issä annettuja tiedostoja. `Move`-parametri kuten edellä.

`function GetComponent(index: integer) : TProjectComponent;`

Funktio palauttaa järjestyksessä `index`-luvun osoittaman komponentin, joka on luotu konstruktorissa.

`TVCmdProjectDetachComponent = class(TVCmdProject)`

Komento komponentin irrottamiseksi projektista.

`constructor Create(comp : TProjectComponent);`

Parametrina annetaan irrotettava komponentti.

`property component : TProjectComponent read fcomponent;`

Ominaisuuden avulla voidaan saada selville irrotettava komponentti.

`TVCmdProjectMoveComponent = class(TVCmdProject)`

Komento komponentin siirtämiseksi projektin vaiheesta tai projektista toiseen.

`constructor Create(comp : TProjectComponent; target: TProjectContainerItem);`

Parametrina annetaan siirrettävä komponenti, sekä kohdesolmu, jonka alaisuuteen komponentti siirretään.

`property component : TProjectComponent read fcomponent;`

Ominaisuuden avulla voidaan saada selville siirrettävä komponentti.

`property target : TProjectContainerItem read ftarget;`

Ominaisuuden avulla voidaan saada selville komponentin uusi omistaja.

`TVCmdProjectActivatePhase = class(TVCmdProject)`

Komento projektin vaiheen aktivoimiseksi. Käskee `WndManageria` asettamaan oikean työpöydän, eli näyttämään vain uuteen vaiheeseen kuuluvat sovellukset.

`constructor Create(phase: TProjectPhase);`

Parametrina annetaan aktivoitava vaihe.

`property phase : TProjectPhase read fphase;`

Ominaisuuden avulla saadaan selville aktivoitu vaihe.

`TVCmdProjectActivateProject = class(TVCmdProject)`

Komento tietyn projektin aktivoimiseksi. Sulkee edelliseen projektiin liittyvät sovellukset, mikäli käyttäjä ei peru toimintoa.

`constructor Create(projectrootitem: TProjectRoot);`

Parametrina annetaan aktivoitavan projektin juurisolmu.

`property project : TProject read fproject;`

Ominaisuuden avulla saadaan selville aktivoitava projekti.

`TVCmdProjectActivateComponent = class(TVCmdProject)`

Komento tietyn komponentin aktivoimiseksi. Palauttaa sovelluksen (komennolla `gui.restore`) jos komponentti on jo auki sovelluksessa, muutoin käynnistää uuden sovelluksen komponentin tiedostonimen perusteella.

`constructor Create(comp: TProjectComponent);`

Parametrina annetaan aktivoitava komponentti.

`property component : TProjectComponent read fcomponent;`

Ominaisuus aktivoitavan komponentin selville saamiseksi.

`TVCmdProjectSave = class(TVCmdProject)`

Komento aktiivisen projetin tallettamiseksi.

`TVCmdProjectSaveAs = class(TVCmdProject)`

Komento aktiivisen projektin tallettamiseksi tietylle nimelle.

`constructor Create(filename: string);`

Parametrina annetaan luotavan projektititedoston nimi.

`TVCmdUpdateServices = class(TVCmdUpdate)`

Komento, joka päivittää Verstaan valikkoon alisovellusten tarjoamat palvelut, eli listan niistä palveluista, joita käyttäjä voi käynnistää. Hakee `AppManagerilta` listan kaikista palveluista, ja kutsuu `formMainin` `ClearServices`- ja `AddService`-metodeita.

`TVCmdUpdateWorkdir = class(TVCmdUpdate)`

Siirtyy aktiivisen projektin työhakemistoon komennolla `ChDir`. Näyttää virheilmoituksen, mikäli siirtyminen ei onnistu.

`TVCmdNotifyProjectContents = class(TVCmdNotify)`

Ilmoitus siitä, että projektin sisältö on muuttunut.

`TVCmdNotifyProjectState = class(TVCmdNotify)`

Ilmoitus siitä, että projektin tila (auki tai kiinni) on muuttunut.

`constructor Create(open: boolean);`

Parametrina tieto uudesta tilasta.

`property open: boolean read fopen;`

Ominaisuus uuden tilan selville saamiseksi.

`TVCmdNotifyProjectPhase = class(TVCmdNotify)`

Ilmoitus siitä, että projektin vaihe on muuttunut.

`TVCmdNotifyUser = class(TVCmdNotify)`

Komento, jonka avulla käyttäjälle voidaan näyttää viestejä. Kutsuu `formMainin` `ShowInfo`-metodia.

`constructor Create(msg: string);`

Parametrina annetaan näytettävä viesti.

`TVCmdWndsNew = class (TVCmd)`

Komento uuden ikkunan liittämiseksi Verstaan tietoisuuteen.

`constructor Create(wnd: TWnd); overload;`

Konstruktorin avulla voidaan ottaa käyttöön muualla luotu `TWnd`-olio.

`constructor Create(form: TForm); overload;`

Konstruktorin avulla voidaan luoda lomakkeesta `TVWnd`-olio.

`TVCmdWndsClose = class (TVCmd)`

Komento ikkunan poistamiseksi Verstaasta.

`constructor Create(wnd: TWnd); overload;`

Konstuktorin parametrina poistettava ikkuna.

`constructor Create(form: TForm); overload;`

Konstruktorin parametri lomake, jota vastaava ikkuna tulee poistaa.

`TVCmdWndsActivate = class (TVCmd)`

Komento tietyn ikkunan aktivoimiseksi.

`constructor Create(wnd: TWnd); overload;`

Parametrina aktivoitava ikkuna.

`constructor Create(tag: Integer); overload;`

Parametrina aktivoitavan ikkunan kokonaislukutunniste.

`TVCmdWndsMinimize = class(TVCmdWnds)`

Komento kaikkien ikkunoiden minimoimiseksi.

`TVCmdWndsRestore = class(TVCmdWnds)`

Komento minimoitujen ikkunoiden palauttamiseksi.

`TVCmdWndsTile = class(TVCmdWnds)`

Komento ikkunoiden järjestämiseksi vierekkäin.

`TVCmdWndsCascade = class(TVCmdWnds)`

Komento ikkunoiden järjestämiseksi limittäin.

Tiedostokomennot

Tiedostokomennot luodaan ja suoritetaan kun alisovellus on ilmoittanut käsitelleensä tiettyä tiedostoa. Kommennot käskevät vastaavan `TApp`-olion päivittää tietonsa kyseisen tiedoston osalta. Komentojen konstruktorin parametreina annetaan komennon lähettänyt sovellus sekä käsitelty tiedosto.

`TVCmdFileOpened = class (TVCmdFile)`

Suoritetaan, kun sovellus on avannut uuden tiedoston.

`TVCmdFileSaved = class (TVCmdFile)`

Suoritetaan kun sovellus on tallentanut tiedoston.

`TVCmdFileClosed = class (TVCmdFile)`

Suoritetaan, kun sovellus on sulkenut tiedoston.

`TVCmdFileModified = class (TVCmdFile)`

Suoritetaan, kun sovellus on muokannut tiedostoa.

7.31 VWnd.pas

Sisältää luokan, joka kuvaa Verstaan omia ikkunoita.

`type TVWnd = class (TPWnd)`

Luokka kuvaa sellaista ikkunaa, johon liittyy lomake (`TForm`). Ikkuna hoitaa siis paikkaan ja kokoon liittyvät toiminnot kutsumalla lomakkeen vastaavia metodeja. Tätä tyyppiä ovat Verstasympäristössä kaikki Verstaan omat ikkunat, kuten pääikkuna ja etsintälomake.

`function GetParent: TProjectContainerItem; override;`

Palauttaa projektiin kuuluvan komponentin, jonka alaisuuteen ikkuna kuuluu. Verstaan ikkunoiden tapauksessa sellaista ei ole, joten palauttaa `nil`.

`constructor Create(form: TForm); reintroduce;`

Konstruktorille annetaan lomake, jota käsittelemällä ikkuna toteuttaa yleisen ikkunan hallinnointikomennot, kuten minimoinnin sekä koon ja paikan asettamisen.

`procedure Minimize; override;`

`procedure Hide; override;`

`procedure Show; override;`

`procedure Restore; override;`

`procedure SetGeometry(X, Y, width, height: integer); override;`

Edelliset kts. TWnd.

`property form: TForm read fform;`

Ominaisuus, jonka avulla saadaan selville ikkunaa vastaava lomake.

7.32 Wnd.pas

Sisältää yleistä ikkunaa kuvaavan luokan.

`TWnd = class`

Yleistä ikkunaa kuvaava abstrakti luokka. `WndManager` käsittelee kaikkia ikkunoita `TWnd`-tyyppisinä, jolloin sen ei tarvitse ottaa kantaa siihen, onko kyseessä alisovelluksen ikkuna voi Verstaan oma ikkuna.

`function GetDesktop: integer; virtual; abstract;`

Palauttaa kokonaislukuna työpöydän, jolle ikkunan kuuluu.

`function GetShowInAll: boolean; virtual; abstract;`

Palauttaa tiedon siitä, tuleeko ikkuna näyttää kaikilla työpöydillä.

`procedure Minimize; virtual; abstract;`

Minimoi ikkunan.

`procedure Hide; virtual; abstract;`

Piilottaa ikkunan.

`procedure Show; virtual; abstract;`

Näyttää ikkunan.

`procedure Restore; virtual; abstract;`

Palauttaa minimoidun ikkunan.

`procedure SetGeometry(X, Y, width, height: integer); virtual; abstract;`

Asettaa ikkunan paikan ja koon annettujen parametrien mukaisesti.

`property caption : string read fcaption write fcaption;`

Ikkunan otsikko.

`property minimizable:boolean read fminimizable write fminimizable;`

Ominaisuus, jolla voidaan selvittää saako ikkunan minimoida.

`property showInAll: boolean read GetShowInAll write fshowInAll;`

Ominaisuus, joka kertoo tuleeko ikkuna näytää kaikilla työpöydillä.

`property desktop : integer read GetDesktop;`

Palauttaa kokonaislukuna ikkunan työpöydän.

`property tag : integer read ftag write ftag;`

Ikkunan uniikki tunniste Verstaan sisällä. Käytetään valikossa erottamaan ikkunat toisistaan.

7.33 WndManager.pas

Sisältää ikkunoiden hallinnoimiseen tarvittavat luokat.

`TWndManager = class`

Ikkunoinnin hallinnasta vastaava luokka. Luokka on singleton, jota kutsutaan lähinnä ikkunoiden järjestämiskomentoja vastaavista `TVCmd`-luokasta perityistä luokista. Manageri pitää kirjaa kaikista Verstaan toimintaan liittyvistä graafisista ikkunoista, ja toteuttaa niiden järjesteämiseen käytettävät komennot, kuten Tile ja Cascade.

`procedure AddWnd(window: TWnd);`

Lisää ikkunan managerin tietoon. Ikkuna on yleistä `TWnd`-tyyppiä, eikä manageri ota kantaa siihen onko kyseessä Verstaan oma ikkuna vai TCP:n yli käskytettävä alisovelluksen ikkuna.

`procedure RemoveWnd(window: TWnd);`

Poistaa ikkunan managerin tietorakenteesta. Ikkunaa ei tämän jälkeen enää oteta huomioon järjestelytoimenpiteissä.

`function GetWnd(tag: integer) : TWnd;`

Palauttaa ikkunan sille luonnin yhteydessä annetun yksikäsitteisen kokonaislukuarvon perusteella.

`procedure SetDesktop(desktop: integer);`

Vaihtaa virtuaalista työpöytää. Toiminto on toteutettu siten, että manageri käyttää ikkunan `desktop`-ominaisuutta saadakseen selville kuuluuko ikkuna halutulle työpöydälle, ja sen mukaan joko näyttää tai piilottaa ikkunan.

`procedure MinimizeAll;`

Minimoi kaikki ikkunat, joiden `minimizable`-arvo on tosi. Tämä siksi, että on mahdollista olla ikkunoita, joita toiminto ei koske (esimerkiksi Verstaan oma pääikkuna).

`procedure RestoreAll;`

Palautaa kaikki ikkunat lähettämällä niille `restore`-komennon.

`procedure Tile;`

Järjestää ikkunat työpöydittäin vierekkäin siten, että kaikilla työpöydillä näkyvät ikkunat ovat aina samoilla paikoilla, ja loppujen kesken tila jaetaan tasan sen mukaan, paljonko ikkunoita on yhdellä työpöydällä enintään. Katso myös Ideoita jatkokehittäjälle -luku.

`procedure Cascade;`

Järjestää ikkunat limittäin, siten että ainakin osa kaikista ikkunoista on näkyvillä. Toiminto tehdään työpöydittäin.

`procedure GetWindows(var list:TList);`

Palauttaa parametrina annettussa listassa osoittimet kaikki managerin tiedossa oleaviin ikkunoihin.

8 Verstas Solver -sovelluksen luokat käännösyksiköittäin

8.1 Main.pas

`TClientThread = class(TThread)`

TCP-asiakkaan säikeestä vastaava luokka.

`procedure TerminateClient();`

Synkronoidusti kutsuttava metodi, joka lopettaa säikeen, jos yhteys palvelimeen menetetään.

`procedure Send;`

Synkronoidusti kutsuttava metodi, joka kutsuu palvelimelta tulevaa dataa käsittelevää metodia.

`procedure Parse(data: String);`

Erottelee luetusta datasta komennon ja parametrit.

`constructor Create(form: TFormSolver); virtual;`

Konstruktori.

`procedure Execute; override;`

Metodi, jota kutsutaan säikeen käynnistyessä. Lukee silmukassa palvelimelta tulevaa dataa.

`TConsoleAppThread = class(TThread)`

`procedure ExecApp();`

Suorittaa ulkoisen ohjelman se. standardisyöttövirrat ohjataan teksti-ikkunaan, ja ulkoisen ohjelman ilmoitukset luoduista tiedostoista lähetetään palvelimelle.

`procedure ReadBuffer();`

Synkronoidusti kutsuttava metodi, joka kirjoittaa ulkoisen ohjelman tulosteen teksti-ikkunaan.

`procedure ReadDataBuffer();`

Synkronoidusti kutsuttava metodi, joka lähettää palvelimelle ilmoituksen ulkoisen ohjelman luomista tiedostoista.

`procedure UpdateGUI();`

Synkronoidusti kutsuttava metodi, joka päivittää pääikkunan graafisia komponentteja.

`constructor Create(form: TFormSolver); virtual;`

Konstruktori.

`destructor Destroy; override;`

Destruktori.

`procedure Execute; override;`

Metodi, jota kutsutaan säikeen käynnistyessä. Kutsuu `ExecApp`-metodia.

`procedure SetCommand(command: String; param: String);`

Asettaa suorittettavan ohjelman nimen ja parametrit.

`function GetPid: integer;`

Palauttaa suoritettavan ohjelman prosessitunnuksen.

`procedure SetPid(pid: integer);`

Tallentaa suoritettavan ohjelman prosessitunnuksen muuttujaan.

`TformSolver = class(TForm)`

Ohjelman pääluokka.

`function GeometryAsString() : String;`

Palauttaa pääikkunan koon ja sijainnin merkkijonona.

`function GetArgsAsStr(): String;`

Palauttaa merkkijonona komentoriviparametrit, joilla ohjelman pääkkunan tila (koko, paikka, pienennetty) voidaan palauttaa.

`function GetCheckedRButton(radioGroup: TRadioGroup) : String;`

Palauttaa merkkijonona valitun radiopainikkeen tekstin.

`function CheckCmdLine() : boolean;`

Erottelee saadut komentoriviparametrit ja toimii niiden perusteella. Jos komentoriviparametri on tunnettu, palautetaan tosi, muutoin epätosi.

`procedure ReadConfig();`

Lukee asetustiedoston.

`procedure WriteConfig();`

Kirjoittaa asetustiedoston.

`procedure SetRadioGroup(section: String; item: String; radioGroup: TRadioGroup);`

Asettaa radiopainikeryhmän asetustiedoston perusteella.

`procedure ArrangeComponents();`

Järjestää pääikkunan komponentit.

`procedure tcpClientConnected(Sender: TObject);`

Lähettää ohjelman tunnisteen ja tuetut rajapinnat palvelimelle TCP-asiakkaan käynnistyessä.

`procedure SetInterfaces();`

Asettaa tuetut rajapinnat merkkijonolistaan.

`procedure ReceiveData(data: String; param: String);`

Käsittelee palvelimelta vastaanotetut rajapintakomennot.

`procedure SendData(data: TAppManInterface); overload;`

Ylikuormitettava metodi, joka lähettää palvelimelle rajapintakomennon.

`procedure SendData(data: TAppManInterface; param: String); overload;`

Ylikuormitettava metodi, joka lähettää palvelimelle rajapintakomennon parametreilla.

`procedure SetGeometry(geometry: String);`

Asettaa pääikkunan koon ja sijainnin.

`procedure StopSolve(forced: boolean);`

Pysäyttää laskentaprosessin hallitusti tai pakotetusti.

`procedure ControlState(state: boolean); overload;`

Ylikuormitettava metodi, joka asettaa komponentin tilan (valittavissa/ei valittavissa).

`procedure ControlState(exclude: TComponent; state: boolean); overload;`

Ylikuormitettava metodi, joka asettaa komponentin tilan (valittavissa/ei valittavissa) poislukien parametrina annettu komponentti.

`procedure FormResize(Sender: TObject);`

Pääikkunan koon muuttuessa kutsuttava metodi, joka järjestää ikkunan komponentit uudelleen.

`procedure ExecuteApplication(command: String; param: String); overload;`

Ylikuormitettava metodi, joka luo `TConsoleAppThread`-säikeen ja käynnistää sillä komennon parametreilla.

`procedure ExecuteApplication(command: String); overload;`

Ylikuormitettava metodi, joka luo `TConsoleAppThread`-säikeen ja käynnistää sillä komennon ilman parametreja.

`procedure FormDestroy(Sender: TObject);`

Destruktori.

`procedure buttonRunKeyDown(Sender: TObject; var Key: Word;`

Run-painikkeen käyttäytyminen painettaessa näppäintä.

`procedure buttonRunKeyUp(Sender: TObject; var Key: Word;`

Run-painikkeen käyttäytyminen vapautettaessa näppäin.

`procedure buttonRunClick(Sender: TObject);`

Käynnistää, pysäyttää tai lopettaa laskentaprosessin Run-painiketta painettaessa.

`procedure buttonMakeClick(Sender: TObject);`

Käynnistää `make`-ohjelman Make-painiketta painettaessa.

`procedure buttonCreateMakeClick(Sender: TObject);`

Käynnistää `mkmf`-ohjelman Create Makefile -painiketta painettaessa.

`procedure copyMenuItemClick(Sender: TObject);`

Kopioi teksti-ikkunan valinnan leikepöydälle.

`procedure clearMenuItemClick(Sender: TObject);`

Tyhjentää teksti-ikkunan.

`procedure selectAllMenuItemClick(Sender: TObject);`

Valitsee teksti-ikkunan sisällön kokonaisuudessaan.

`procedure radioGroupLibsClick(Sender: TObject);`

Suorittaa `make clean` -komennon vaihdettaessa kirjastoa radiopainikkeella.

`procedure exitMenuItemClick(Sender: TObject);`

Poistuu sovelluksesta.

`procedure aboutMenuItemClick(Sender: TObject);`

Avaa ohjelman tietoja näyttävän ikkunan.

`procedure FormCloseQuery(Sender: TObject; var CanClose: Boolean);`

Esittää käyttäjälle varmistuskysymyksen sovellusta suljettaessa, mikäli käynnissä on prosessi.

`procedure solverHelpMenuItemClick(Sender: TObject);`

Avaa avustustoiminnon.

`function GetDir() : String;`

Palauttaa merkkijonona työhakemiston.

`function SetDir(dir: String) : boolean;`

Asettaa työhakemiston.

`procedure SetFileAge(age: integer);`

Asettaa tiedoston aikaleiman muuttujaan.

`function GetFileAge(): integer;`

Palauttaa tiedoston aikaleiman.

`procedure SetRunning(state: boolean);`

Asettaa ulkoisen ohjelman suoritus käynnisssä -lipun.

`constructor Create(AOwner: TComponent); override;`

Konstruktori.

8.2 Interfaces.pas

`Application`- ja `AppManager`-rajapintojen tyyppimääritykset.

8.3 Constants.pas

Yleisiä vakioita.

8.4 About.pas

`TformAboutBox = class(TForm)`

Ohjelman tiedot näyttävä lomake.

9 Käyttöliittymä

Tässä luvussa esitellään Verstas- ja Verstas Solver -sovelluksen käyttöliittymät.

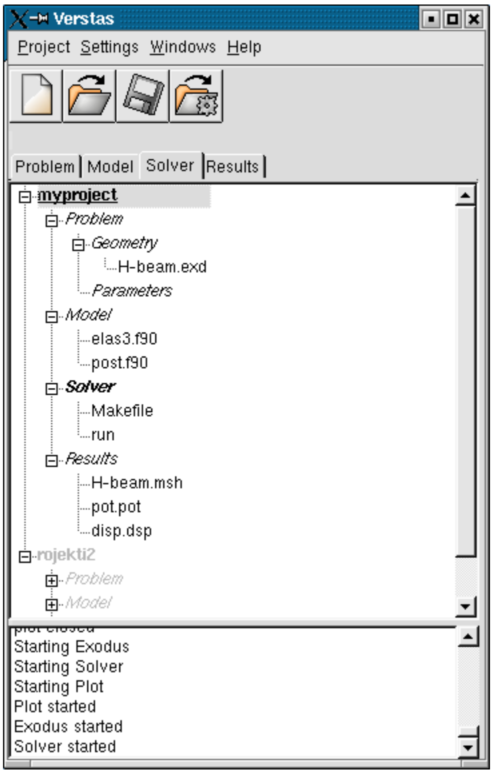
9.1 Verstas

9.2 Pääikkuna

Verstas-sovelluksen pääikkuna (Kuva 8) muodostuu projektin puunäkymästä, välilehdistä, valikoista, työkalupalkista painikkeineen sekä teksti-ikkunasta. Puunäkymä kuvaa projektia ja siinä näkyvät hierarkisesti projektin kuhunkin mallinusvaiheeseen ja alivaiheeseen liitetyt mallinnuskomponentit. Projekteja voi olla avoinna useita yhtäaikaa, joista yksi on kerrallaan aktiivinen. Mallinnuskomponentteja voi kopioida toisesta projektista aktiiviseen projektiin.

Välilehdillä siirrytään mallinnusvaiheesta toiseen, aktiivisessa vaiheessa ovat näkyvissä vain siihen kuuluvat alisovellukset.

Teksti-ikkunassa näkyvät tiedot alisovellusten käynnistymisestä ja sulkeutumisesta sekä niiden tallentamista tiedostoista.

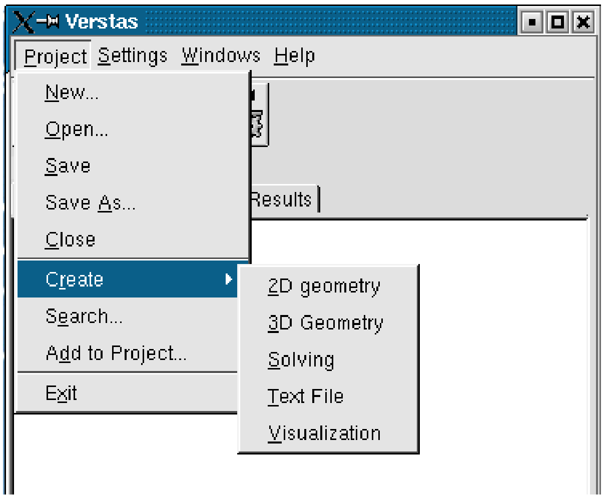


Kuva 8: Verstas-sovelluksen pääikkuna

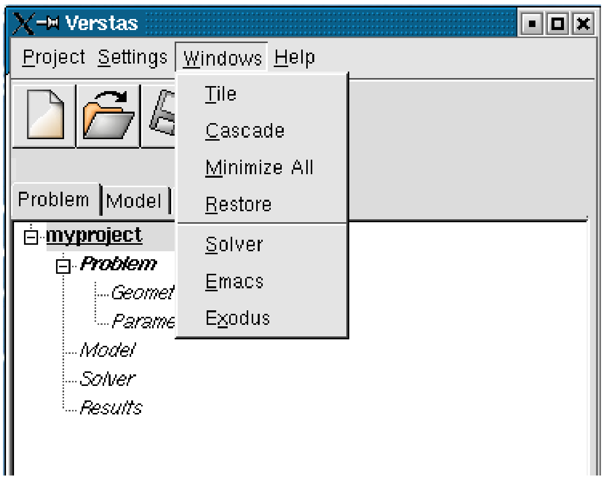
9.2.1 Valikot

Project-valikosta (Kuva 9) voidaan luoda uusi projekti, avata olemassa oleva projekti, tallentaa projekti, sulkea projekti, lisätä projektiin uusi komponentti, käynnistää alisovelluksia tai hakutoiminto sekä sulkea sovellus.

Settings-valikosta käynnistetään käyttäasetusten dialogi. Windows-valikosta (Kuva 10) hallitaan ja aktivoidaan alisovellusten ja Verstaan ikkunoita, joita voidaan järjestää limittäin ja vierekkäin sekä pienentää ja palauttaa. Help-valikosta avataan avustustoiminto ja ohjelman tiedot näyttävä ikkuna.



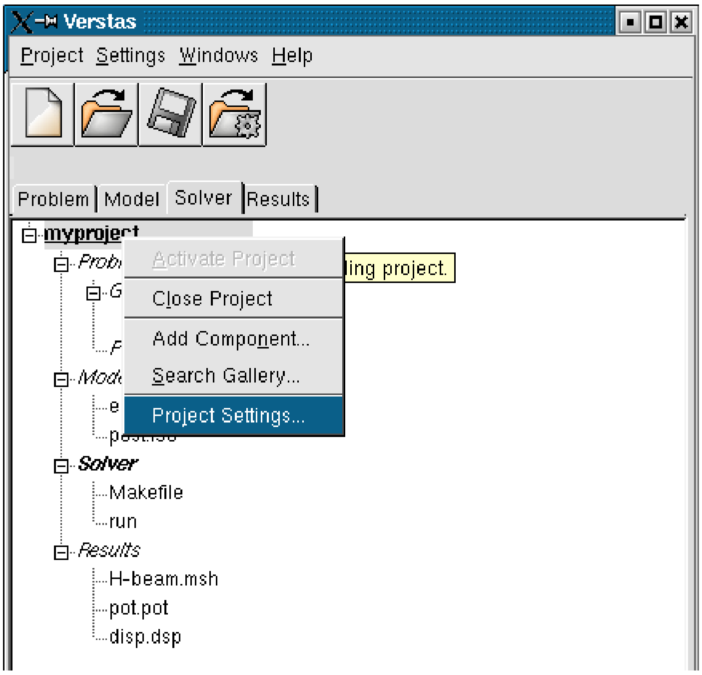
Kuva 9: Project-valikko



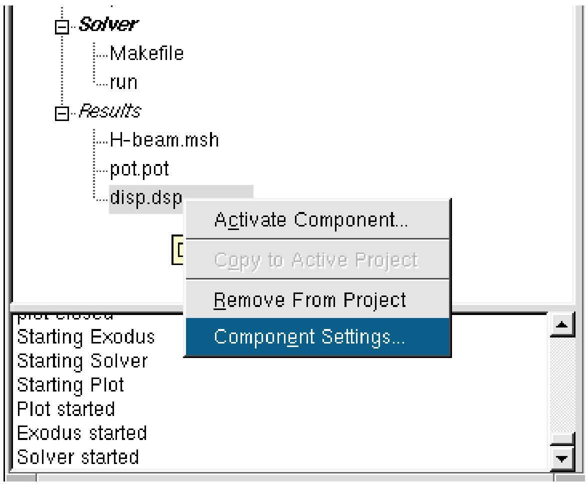
Kuva 10: Windows-valikko

Projektin ponnahdusvalikosta (Kuva 11) voidaan aktivoida ja sulkea projekti, lisätä mallinnuskomponentti, käynnistää hakutoiminto tai siirtyä projektiasetuksiin.

Mallinnuskomponentin ponnahdusvalikosta (Kuva 12) voidaan aktivoida valittu komponentti sitä vastaavaan alisovellukseen, kopioida komponentti aktiiviseen projektiin, poistaa komponentti projektista tai siirtyä komponentin asetuksiin.



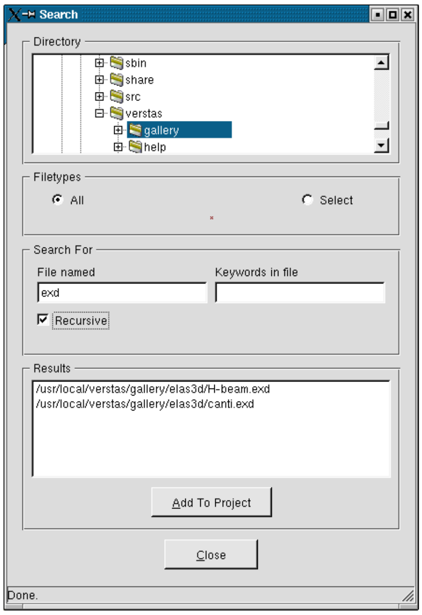
Kuva 11: Projektin ponnahdusvalikko



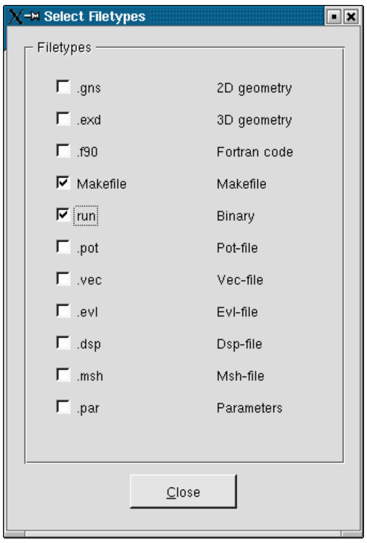
Kuva 12: Mallinnuskomponentin ponnahdusvalikko

9.2.2 Etsintädialogi

Etsintädialogilla (Kuva 13) suoritetaan hakuja ja lisätään haun tuloksena saatuja mallinnuskomponentteja projektiin. Haku suoritetaan tiedostonimeen ja/tai tiedostoon liittyvien avainsanojen perusteella joko kaikentyyppisille tiedostoille tai erillisellä dialogilla (Kuva 14) valittaville tiedostotyypeille.



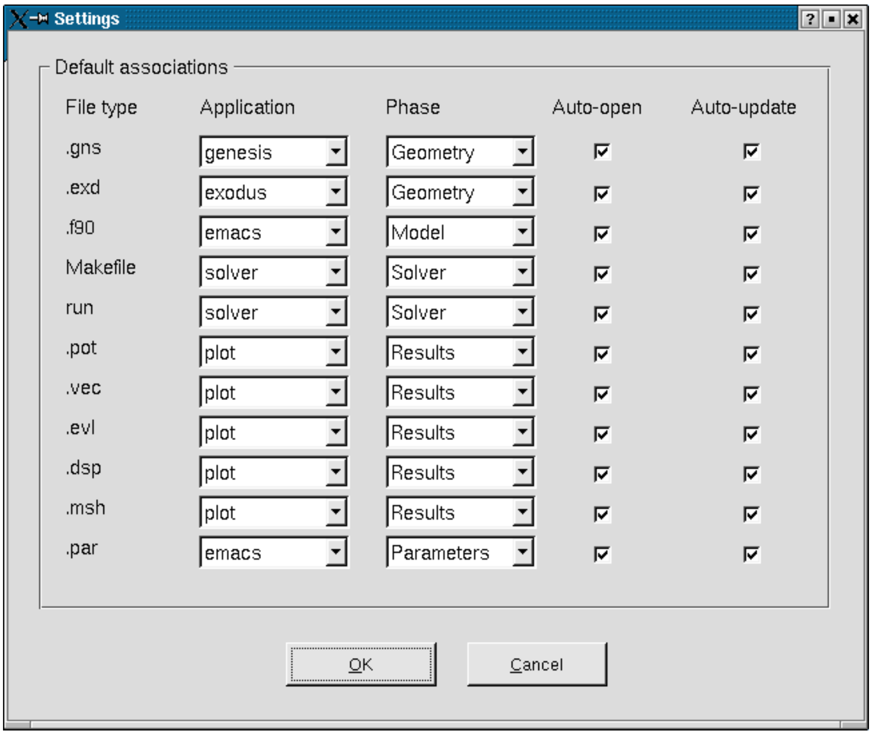
Kuva 13: Etsintädialogi



Kuva 14: Tiedostotyyppien valintadialogi

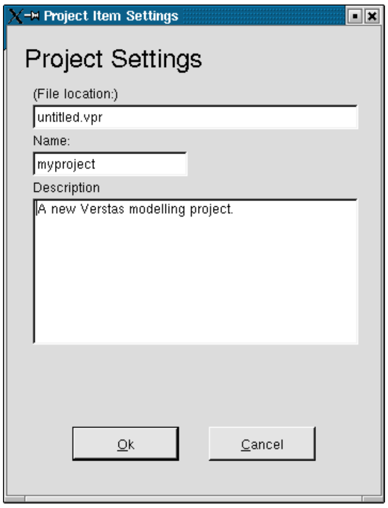
9.2.3 Asetusdialogit

Käyttäjäasetusten dialogilla (Kuva 15) valitaan tiedostotyyppiä vastaava alisovellus sekä mallinnusvaihe, johon kukin alisovellus kuuluu. Myös automaattinen avaus alisovellukseen ja automaattinen päivitys alisovelluksissa tiedoston muuttuessa voidaan valita kullekin tiedostotyypille.

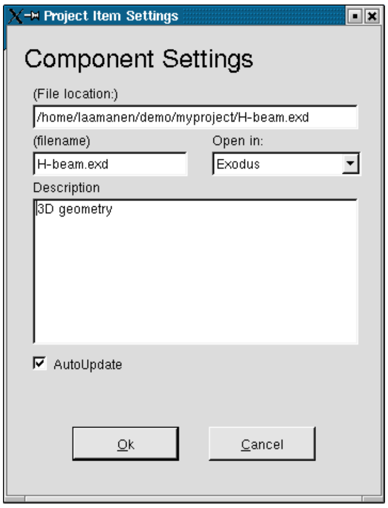


Kuva 15: Käyttäjäasetuksien dialogi

Projektin asetustusdialogilla (Kuva 16) asetetaan projektin nimi ja kuvaus. Mallinnuskomponentin asetusdialogilla (Kuva 17) valitaan komponenttia vastaava alisovellus, asetetaan komponentille kuvaus ja valitaan automaattinen päivitys alisovelluksissa.



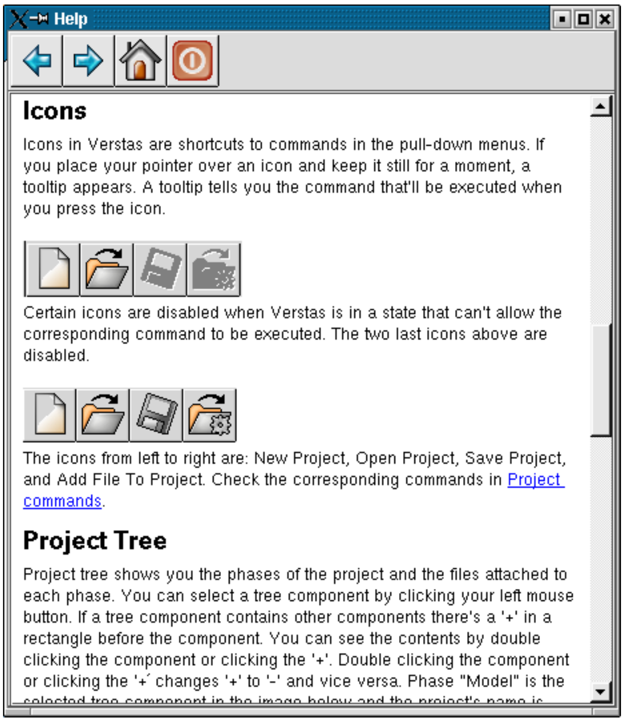
Kuva 16: Projektin asetusdialogi



Kuva 17: Mallinnuskomponentin asetuksien dialogi

9.2.4 Avustustoiminto

Avustustoiminto (Kuva 18) näyttää sovelluksen avustusdokumentteja. Painikkeilla voidaan siirtyä dokumenttihistoriassa edelliseen ja seuraavaan dokumenttiin tai ensimmäiseen dokumenttiin. Avustustoimintoa käytetään myös Verstas Solver -sovelluksessa.



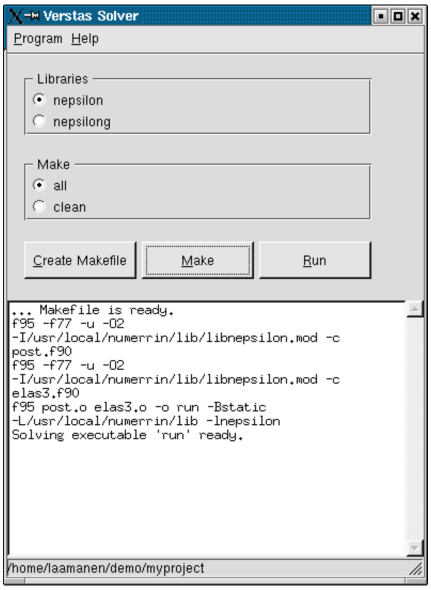
Kuva 18: Avustustoiminto

9.3 Verstas Solver

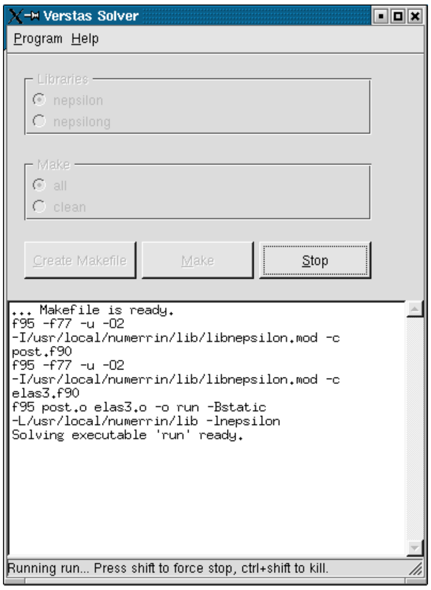
Verstas Solver -sovelluksella (Kuva 19) hallitaan kääntämistä ja laskentaa. Program-valikosta suljetaan sovellus, Help-valikosta käynnistetään avustustoiminto.

`Makefile`-tiedosto luodaan Create Makefile -painikkeella Libraries-kohdasta valitun kirjaston mukaisesti. `Make`-ohjelma käynnistetään Make-painikkeellla (all: kääntäminen, clean: käännöksessä syntyneiden tiedostojen poistaminen).

Käännetty laskentaohjelma käynnistetään Run-painikkeesta, joka muuttuu laskennan ajaksi Stop-painikkeeksi, jolla laskentaprosessi voidaan tarvittaessa keskeyttää hallitusti tai pakotetusti tai lopettaa (Kuva 20).



Kuva 19: Verstas Solver -sovellus



Kuva 20: Laskentaprosessi käynnissä

10 Ideoita jatkokehittäjälle

- Verstaan käännösyksiköihin jäi kiireen takia paljon ristiviittauksia toisiinsa. Näistä on mahdollista päästä eroon luomalla alkuperäisen suunnitelman mukaisesti Verstaan sisäisiä komentoja.

- Mahdollisuus tallettaa kaikki muutokset kaikissa sovelluksissa napin painalluksella jäi toteuttamatta. Tämä, kuten muitakin uusia toiminnallisuuksia, voidaan toteuttaa Verstaan sisäisenä komentona.

- Epämääräisistä viiveistä `AppManager`in `LaunchAndOpen`-metodissa ja `TAppin` `VCmdFileSaveHandler`issä tulisi päästä eroon. Tässä voitaisiin käyttää `CmdParserin` `WaitFor`-metodia, mutta menetelmää ei projektin puitteissa ehditty riittävästi testaamaan.

- Sisäisessä tallennuksessa komponenttien polut voisivat olla suhteellisia projektihakemistoon nähden. Projekti olisi tällöin jopa mahdollista siirtää eri paikkaan.

- Indy:n TCPServer-komponentti on huono - mm. komponentin vapauttaminen siististi ei onnistu, eikä sitä käynnistyksen jälkeen voida asettaa pois päältä. Komponentti olisi mahdollisesti korvattavissa kirjoittamalla itse vastaava toiminnalisuus.

- `Libc`:tä käyttävissä kohdissa tulisi toteuttaa vastaava toiminnalisuus myös Windows-alustalle.

- Sovelluksissa auki olevat tiedostot voitaisiin tallettaa viittauksina projektin komponentteihin, joiden ei tarvitsisi sisältää absoluuttista polkua.

- Solver-sovelluksesta voisi tehdä modulaarisemman. Verstaan ja Solverin välille voisi toteuttaa tiedonsiirron siten, että Solveria voisi tehokkaasti käyttää toiselta koneelta. Etäkäyttöä ajatellen olisi ehkä parempi, että laskentaohjelma olisi komentorivipohjainen (vaativan laskennan voisi jättää käyntiin esim. screen-ohjelmaan) ja sitä hallittaisiin ja seurattaisiin Verstaan käyttöliittymästä.

- Solver-sovelluksen ja laskentaohjelmien kommunikaatiota voisi muuttaa siten, että laskentaohjelmat lukisivat `.stopped`- ja `.mesh`-tiedostojen sijaan standardisyöttövirtaa. Lisäksi Solveria voisi kehittää siten, että `Makefile`a luotaessa siihen sisällytettäisiin ainoastaan sellaiset fortran-koodit, jotka kuuluvat projektiin.

- Tile-komentoa olisi mahdollista optimoida siten, että kaikki tyhjä tila käytettäisiin hyödyksi jokaisella työpöydällä. Kaikilla työpöydillä näkyvät ikkunat jakaisivat siis oman osansa näytöstä, ja muu osa jaettaisiin työpöydittäin parhaalla mahdollisella tavalla.

- Ikkunoilla on `showInAll`-ominaisuus, jonka perusteella ikkunat voitaisiin asettaa näkymään kaikissa vaiheissa. Käyttäjä ei kuitenkaan voi määrätä, onko ominaisuus käytössä.

- ELisp-moduulin jatkokehitysohjeita on mainittu lähdekoodissa.

11 Yhteenveto

Tässä dokumentissa kuvattiin Verstas-projektin sovellusten toteutusta. Raportissa on kuvattu Verstas- ja Verstas Solver -sovellusten arkkitehtuuria, rajapintoja, toiminnallisuutta sekä käyttöliittymää. Toteutusta on esitelty käännösyksiköittäin siten, että kunkin yksikön kuvaus sisältää yksikön sisältämien luokkien ja niiden metodien esittelyt sekä sisäiset algoritmit ja tietorakenteet oleellisilta osiltaan.

Viitteet

[1] Becks, Ari, *Delphi - sovelluksentekijän opas*, Gummerus kirjapaino Oy, Jyväskylä, 1995.

[2] Cantù, Marco, *Mastering Delphi 5*, 1999.

[3] Gamma, E, Helm, R, Johnson, R, Vlissides, J, *Design Patterns - Elements of Reusable Object-Oriented Software*, 2001.

[4] Santanen Jukka-Pekka, Gradupohja *Microsoft Wordin template-tiedosto ja lyhyt ohje tyylien käyttöön*, saatavilla WWW-muodossa
<URL: `http://www.mit.jyu.fi/progradut/tyylipohjat/gradupohja.dot`>, viitattu 30.10.2002.

[5] Santanen Jukka-Pekka, *Opinnäytteiden kirjoittaminen, lyhyt oppimäärä*, saatavilla WWW-muodossa
<URL: `http://www.mit.jyu.fi/santanen/info/kirjoittamisesta.html`>, viitattu 30.10.2002

[6] Verstas projekti, *Vaatimusmäärittely*, 2002

[7] Verstas projekti, *Käyttötapaukset*, 2002