Suunnitteludokumentti

Koski-ryhmä

Helsinki 18.5.2004

Ohjelmistotuotantoprojekti

HELSINGIN YLIOPISTO
Tietojenkäsittelytieteen laitos

Kurssi

581260 Ohjelmistotuotantoprojekti (6 ov)

Projektiryhmä

Olli Alm
Seppo Hätönen
Sini Ruohomaa
Antti Takalahti
Sampo Yrjänäinen
Arto Åkerlund

Asiakas

Teemu Kerola

Johtoryhmä

Raine Kauppinen (ohjaaja)
Juha Taina (vastuuhenkilö)
Turjo Tuohiniemi (vastuuhenkilö)

Kotisivu

`http://www.cs.helsinki.fi/group/koski`

Versiohistoria

Versio	Päiväys	Tehdyt muutokset
1.0	15.3.2004	Ensimmäinen versio
1.2	18.3.2004	FTR:n perusteella korjattu versio
1.3	1.5.2004	Toteutusvaiheen pohjalta päivitetty versio

Sisältö

1 Johdanto

Koski-projekti on osa Helsingin yliopiston Tietojenkäsittelytieteen laitoksen kurssia 581260 Ohjelmistotuotantoprojekti. Projekti toteutetaan keväällä 2004 projektisuunnitelman mukaisesti.

Koski-projektin tavoitteena on kehittää konekielisimulaattori Helsingin yliopiston Tietojenkäsittelytieteen laitoksen kurssin 581305 Tietokoneen toiminta tarpeita varten. Projektiryhmään kuuluvat Olli Alm, Seppo Hätönen, Sini Ruohomaa, Antti Takalahti, Sampo Yrjänäinen ja Arto Åkerlund. Ohjaajana toimii Raine Kauppinen, vastuuhenkilöt ovat Juha Taina ja Turjo Tuohiniemi. Projektin asiakas on Tietojenkäsittelytieteen laitoksen puolesta Teemu Kerola. Projektin tuloksena syntyvä ohjelmisto julkaistaan GNU Lesser General Public lisenssin alla. (Lisää tietoa lisenssistä osoitteessa `http://www.gnu.org/copyleft/lesser.html`.)

Suunnitteludokumentin tarkoituksena on sitoa vaatimusmäärittelyssä annetut ohjelman vaatimukset tekniseen toteutukseen täsmällisesti. Täsmällisyydellä tarkoitetaan tässä yhteydessä riittävän tarkkaa yksityiskohtien teknistä määrittelyä; suunnitteludokumentin avulla on mahdollista toteuttaa ohjelmisto, joka on keskeisiltä toiminnallisuuksiltaan yksikäsitteinen; riittävän osaamistason omaavan toteuttavan tahon vaikutus tuotteeseen pitäisi olla vähäinen. Suunnitteludokumentti ei sisällä testaussuunnitelmaa, vaan se muodostaa oman dokumenttinsa.

2 Yleiskuva

Termit:

eAssari tarkoittaa Harri Laineen kuvailemaa, myöhemmin toteutettavaa, ohjelmaa, joka mahdollistaa laskuharjoitusten automaattisen tarkistamisen. Ellei erikseen mainita, niin ohjelmistolla tarkoitetaan ohjelmistoa ilman eAssari-käyttöliittymää.

API-termi tarkoittaa tässä dokumentissa eAssari-ohjelmaa varten toteutettavaa rajapintaa.

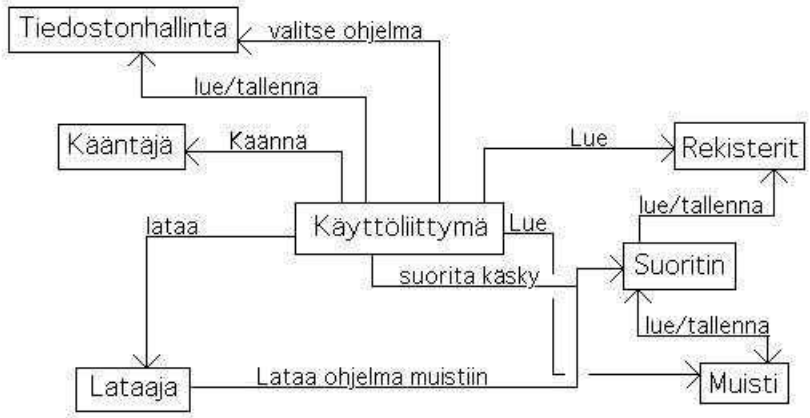
TTK-91 on nimi opetustarkoitukseen spesifioidulle, teoreettiselle koneelle, jota ohjelmalla simuloidaan. Sen konekielestä on aiemmin annettu määrittely, jonka päivitetty versio tomitetaan toteutettavan ohjelman mukana.

.k91-termi tarkoittaa symbolisessa konekielimuodossa olevaa TTK-91-ohjelmaa.

.b91-termillä tarkoitetaan käännettyä, binäärimuotoista TTK-91-ohjelmaa.

Tässä suunnitteludokumentissa kuvatulla ohjelmalla on kaksi käyttöliittymää: konekielisimulaattori ja eAssari. EAssari jää pelkäksi rajapintamäärittelyksi, sen käyttöliittymä toteutetaan myöhemmin. Vaatimusmäärittelydokumentissa esiintyvä Trainer2-nimitys on eAssarin vanha nimi; eAssaria käytetään tästä lähtien, asiakkaalta tulleen tiedon mukaisesti.

Konekielisimulaattorin on määrä korvata Tietojenkäsittelytieteen laitoksen kurssilla Tietokoneen toiminta käytetty TTK-91-simulaattori. Uusi versio mahdollistaa (1) yhteenso-

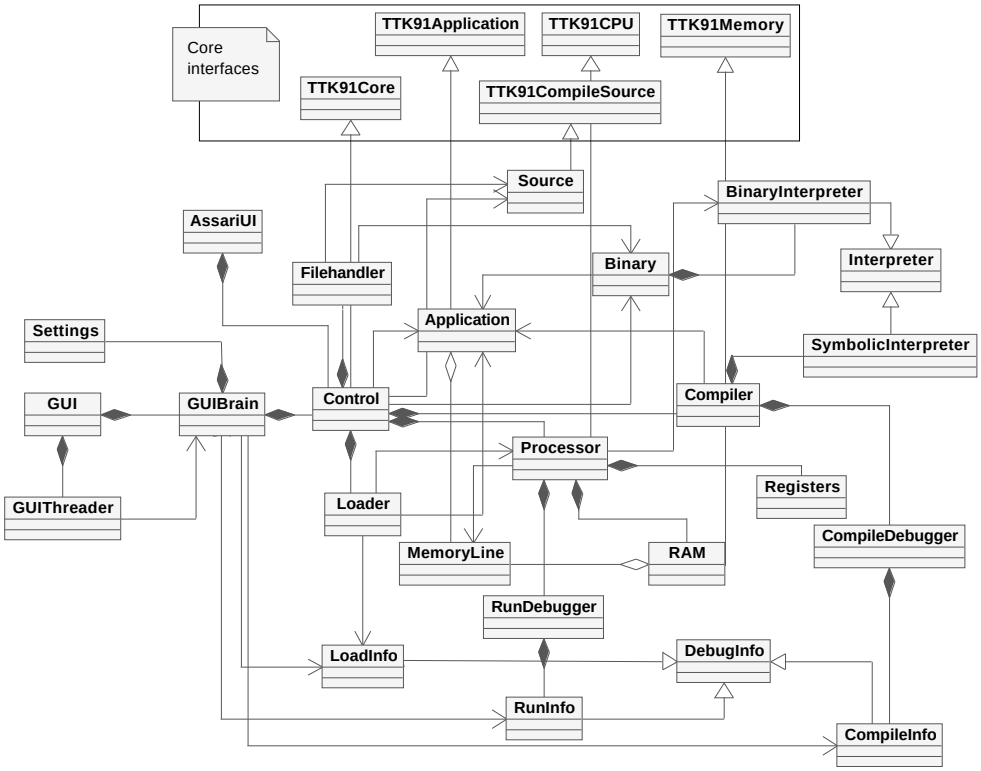


Kuva 1: Ohjelman yleiskuva.

pivuuden aiemman version syöteohjelmien kanssa, (2) konekielisimulaattorin suorituksen sekä Windows- että Linux-käyttöjärjestelmissä ja (3) konekielen suorituksen simuloinnin havainnollistamisen.

Rinnakkaisryhmän kanssa yhdessä sovittu TTK91-rajapinta (valtaosa sen määrittelemistä luokista alkaa merkeillä TTK91) on ennalta määritellyn API-kuvauksen mukainen ohjelmiston rajapinta, joka mahdollistaa myöhemmin liitettävän eAssari-käyttöliittymän tarvitsemat toiminnallisuudet. Toiminnallisuuksia ovat (1) ohjelman kääntäminen ja tilatietojen tulostaminen, (2) ohjelman suoritus ja tilatietojen tulostaminen. Kääntäminen ja suoritus on eAssari-rajapinnassa tarkoitettu tapahtuvan kertaluontoisesti, ilman simulointia. Luokkakaaviossa sivulla 3 on kuvattu rajapintaluokista ne, jotka toteutuvat projektimme luokissa. Rajapintaan kuuluu myös poikkeusluokkia, joita ei tässä erikseen kuvata. Rajapinnan kuvaus on liitteenä.

Kuvassa 1 on kuvattu järjestelmän yleisrakenne ja tärkeimmät palvelut. Varsinainen luokkakaavio seuraa myöhemmin. Käyttöliittymä kontrolloi suoritusta, mutta sen ymmärrys esimerkiksi TTK91-ohjelman suorituksesta on pyritty minimoimaan; kuitenkin animoinnin mahdollistamiseksi tämän minimoinnin mahdollisuudet ovat olleet rajalliset. Tiedostonhallinta toteuttaa mm. levyltä lukemisen ja sinne kirjoittamisen ohjelmaa valittaessa, käännettyä ohjelmaa tallennettaessa ja levy-I/O:ta simuloitaessa. Kääntäjä tulkkaa symbolisen konekäskyn kokonaisluvuksi ja toisaalta kokonaislukuja konekäskyiksi. Palvelut on toteutettu eri luokissa. Lataaja vie käännetyn ohjelman muistiin suorittimen kautta. Suoritin suorittaa käyttöliittymän pyytäessä konekäskyn, ja palauttaa tiedon koneen muuttuneesta tilasta. Muistilla on palvelut tiedon viemiseksi annettuun muistipaikkaan ja muistin sisällön tutkimiseksi. Rekisterit ovat toiminnallisuudeltaan varsin samanlaisia kuin muisti; tiettyyn rekisteriin voidaan tallettaa tietoa ja rekisteriin talletettua tietoa voidaan tarkastella.



Kuva 2: Luokkakaavio

3 Luokkarakenne

Tässä luvussa esitellään luokkarakenne uml-mallinnusta mukaillen yleisellä tasolla. Esitystapa sisältää luokkien keskinäiset suhteet luokkakaavion muodossa sekä lyhyeen kuvauksen jokaisesta luokasta. Kuvauksessa kerrotaan, mistä luokista luokka luodaan, mistä luokista kyseistä luokkaa pääsääntöisesti käytetään ja yleiskuva luokasta. Luokkien käyttötapaukset esitellään luvussa 4, keskeiset toiminnallisuudet luvussa 6. Liite 1 sisältää luokkien Javadoc-kuvaukset.

Luokkakaavio ei sisällä kaikkia luokkia; luokkakaavion selkeyttä on pyritty korostamaan tarkkuuden kustannuksella. Poisjätetyt luokat on esitelty sanallisissa kuvauksissa. Luokkakaaviossa esiintyvät TTK91-luokat viittavat yhteisiin rajapintaluokkiin, joista on sovittu rinnakkaisryhmän kanssa. Erityisesti AssariUI-käyttöliittymärajapinta käyttää kyseisiä rajapintaluokkia.

Animator

- Luodaan GUI-luokasta käyttäjän valitessa animoidun suorituksen.
- Luokka vastaa suoritussyklin eri vaiheiden tarkemmasta havainnollistamisesta animaation avulla. Se saa tarvitsemansa tiedot RunInfo-luokalta, joka sille metodikutsun yhteydessä välitetään.

Application

- Luodaan Binary-luokasta .b91-tiedoston avaamisen yhteydessä tai Compiler-luokasta, kun .k91-tiedoston käännös on valmistunut.
- Käytetään Control-luokasta avaamisen jälkeen.
- Luokka on dataluokka, joka esittää käännettyä ohjelmaa symbolitauluineen, ja huolehtii ohjelman syöte- ja tulostevirroista tallentaen niitä sisäisiin muuttujiinsa sekä mahdollistaen niiden kyselyn.
- Luokan sisältämät ohjelmarivit ovat MemoryLine-olioita, jotka sisältävät komentoa vastaavan muistirivin binääri- ja symbolimuotoisen esityksen.

AssariUI

- Käyttää Control-luokkaa sen TTK91Core-rajapintaluokan kautta.
- Luokka on käyttöliittymä eAssarin rajapintatarpeiden testaukseen. Se kääntää ja suorittaa annetun ohjelman ja tulostaa koneen lopputilan. Luokka luodaan ohjelman käynnistyessä.

Binary

- Luodaan FileHandler-luokasta .b91-tiedostoa avatessa tai Control-luokasta talletettaessa .b91-tiedostoa levylle. Jälkimmäisessä tapauksessa apuna käytetään BinaryInterpreteriä.
- Luokka sisältää ohjelman konekielimuotoisen esityksen. .b91-tiedoston avaamisen yhteydessä se tulkitsee binääriformaattia sekä luo ja palauttaa itsestään Application-olion BinaryInterpreter-luokan avulla. .b91-tiedoston tallentamisen yhteydessä se tulkitsee Application-luokan vastaavasti binääriformaatin mukaiseksi merkkijonoksi, jonka se palauttaa. Binääriformaatti kuvataan esimerkkiohjelman kautta liitteessä 4. Sulkuihin kirjoitetut kommentit eivät kuulu itse formaattiin.

BinaryInterpreter

- Luokka luodaan ja sitä käytetään Binary-luokasta Application-oliota luotaessa ja Processor-luokasta TTK91-ohjelmaa ajettaessa.

- Luokka sisältää hajautustaulun, jonka avulla konekielimuotoinen käsky käännetään symboliseksi ja palveluja tulkitsemiseen.
- Luokka perii Interpreter-luokan.

CompileDebugger

- Luokka luodaan ja sitä käytetään Compiler-luokasta.
- Luokka generoi Compiler-luokan tietojen avulla CompileInfoja, jotka välittävät kääntämisen tilatietoa käyttöliittymälle.
- Luo CompileInfo-olioita, jotka palautetaan Compilerille.

CompileInfo

- Luokka luodaan CompileDebugger-luokasta.
- CompileInfo on dataluokka, joka välittää kääntämisen tilatiedot käyttöliittymälle.
- Käytetään luokasta GUIBrain, joka päivittää sen perusteella GUI-luokan kautta käyttäjän näkymää.

Compiler

- Luodaan ja käytetään Control-luokasta.
- Luokka kääntää rivi kerrallaan .k91-tiedostoa. Se huolehtii syntaksitarkistuksesta, symbolitaulun muodostamisesta sekä konekielikäskyjen generoinnista.

Control

- Käytetään luokista AssariUI ja GUIBrain.
- Control on keskusluokka, joka välittää dataa käyttöliittymän ja muiden luokkien välillä.
- Käyttää luokkia FileHandler, Compiler, Processor, Loader, Binary, Source ja Application.

DebugInfo

- Info-luokkien (LoadInfo, DebugInfo, RunInfo) yläluokka. Sisältää näille yhteiset tilatietomuuttujat.

Filehandler

- Luodaan Control-luokasta.

- FileHandler hoitaa tiedostojen hallintaan liittyviä tehtäviä, mutta ei varsinaisesti tulkitse niiden sisältöä.
- Luo Source- ja Binary-luokat.

GUI

- Luodaan Titokone-luokasta.
- GUI on simulaattorin graafinen käyttöliittymä, josta varsinainen toiminnallisuus on pyritty eristämään GUIBrain-luokkaan ja Animator-luokkaan.
- Käyttää GUIBrain-luokkaa viestien tulkitsemiseen; GUIBrain vastaavasti kutsuu GUI:n päivitysmetodia niiden perusteella. Lisäksi GUI käyttää Animator-luokkaa animointi-ikkunan ja sen muutosten toteuttamiseen.

GUIBrain

- Luodaan GUI-luokasta.
- Luo Control- ja Settings-luokat.
- GUIBrain on graafisen käyttöliittymän apuluokka, joka tulkitsee GUIn vaatimat päivitystiedot asetusten tilan mukaisesti.
- Käyttää luokkia CompileInfo, LoadInfo ja Runinfo Controlin ja Settingsin lisäksi.

GUIHTMLDialog

- Luodaan GUI-luokasta.
- GUIHTMLDialog edustaa About- ja Manaul-dialogeja, jotka lukevat tietonsa HTML-tiedostoista.

GUICompileSettingsDialog

- Luodaan GUI-luokasta.
- GUICompileSettingsDialog edustaa käännösaikaisten asetusten muuttamiseen tarkoitettua dialogia.

GUIRunSettingsDialog

- Luodaan GUI-luokasta.
- GUIRunSettingsDialog edustaa ajoaikaisten asetusten muuttamiseen tarkoitettua dialogia.

GUIThreader

- Luodaan GUI-luokasta.
- GUIThreader on graafisen käyttöliittymän apuluokka, joka auttaa joidenkin GUIBrainin pitkäkestoisempien tehtävien sijoittamisessa omaan Java-säikeeseensä. Sille välitetään GUIBrain-olio, josta se kutsuu varsinaista palvelua käynnistyessään.
- Käyttää luokkaa GUIBrain.

Interpreter

- Luokkien BinaryInterpreter ja SymbolicInterpreter yläluokka, joka sisältää näiden kääntäjien tarvitsemat yhteiset tiedot.

InvalidSymbolException

- Käytetään SymbolTable-luokasta.
- Exception-luokka, joka heitetään kun SymbolTable-luokalta kysellään tuntematto-man symbolin arvoa.

InvalidDefinitionException

- Käytetään SymbolTable-luokasta.
- Exception-luokka, joka heitetään kun SymbolTable-luokalta kysellään tuntematto-man määrittelyn arvoa. Määrittelyt tehdään DEF-valekäskyllä; se eroaa symboleista merkkijono-arvojoukollaan, sillä symbolien arvot ovat aina lukuja.

JTableX

- Käytetään ja luodaan GUI-luokasta.
- JTablen laajennos, joka todettiin tarpeelliseksi käyttöliittymätoteutuksessa.

Loader

- Luodaan Control-luokasta.
- Lataa Application-luokan mukaisen ohjelman prosessorille ja generoi LoadInfo-viestin GUIBrainia varten.
- Käyttää luokkia Processor ja Application sekä luo LoadInfo-luokan.

LoadInfo

- Luodaan Loader-luokasta.
- Käytetään GUIBrain-luokasta.
- LoadInfo on dataluokka, joka sisältää latauksesta johtuneiden tilanmuutosten tiedot.

MemoryLine

- Luodaan luokasta Compiler ja Binary Application-luokkaa tehdessä, luokasta RandomAccessMemory tätä alustettaessa ja Processor-luokasta ajettavan TTK91-konekäskyn kirjoittaessa jotakin muistiin. RandomAccessMemoryssä ja Applicationissa on siis taulukollinen MemoryLinejä, yksi kutakin muistiriviä tai muistiin lataamista odottavaa riviä kohden.
- MemoryLine esittää yhtä ohjelman muistiriviä, ja se sisältää sekä luku- että symbolimuotoisen esityksen siitä.

Message

- Luokka viestien generoimiseen, käytetään missä tahansa luokassa joka luo uuden viestin käyttäjälle näytettäväksi.

Processor

- Luodaan ja käytetään Control-luokasta, lisäksi käytetään Loader-luokasta ja TTK91-CPU-muotoisena AssariUI-luokasta.
- Luokka simuloi ohjelmien suoritusta rivi kerrallaan ja antaa tilatietoa suorituksen etenemisestä RunDebugger-luokalle.
- Processor luo käyttöönsä luokat Registers, RandomAccessMemory sekä RunDebugger, lisäksi käyttää tarvittaessa luomaansa BinaryInterpreter-luokkaa itseään muuttavan koodin symbolisen esityksen mahdollistamiseksi.

RandomAccessMemory

- Luodaan ja käytetään Processor-luokasta sekä käytetään TTK91Memory-muotoisena AssariUI-luokasta.
- Koostuu MemoryLine-olioista.
- Luokka edustaa Processor-luokan suorituksenaikaista muistisisältöä.

Registers

- Luodaan ja käytetään Processor-luokasta.
- Luokka edustaa prosessorin rekistereiden tilaa.

ResourceLoadFailedException

- Luodaan FileHandler-luokasta, otetaan kiinni GUIBrain-luokassa.

- Exception-luokka, kuvaa tilannetta jossa FileHandleriä on pyydetty lataamaan uusi ResourceBundle-luokka tiedostosta ja lataaminen epäonnistuu.

RunDebugger

- Luodaan Processor-luokasta.
- Tulkitsee Processor-luokan antamia tilatietoja ja generoi niiden perusteella GUIBrain-luokalle tilatietoja RunInfo-olioilla.

RunInfo

- Luodaan RunDebugger-luokasta.
- Käytetään GUIBrain-luokasta.
- Tämä dataluokka välittää tietoja prosessorin ajonaikaisesen tilan muutoksista kunkin komennon aikana.

Settings

- Luodaan GUIBrain-luokasta.
- Dataluokka, joka sisältää simulaattorin käyttöliittymän asetukset.

Source

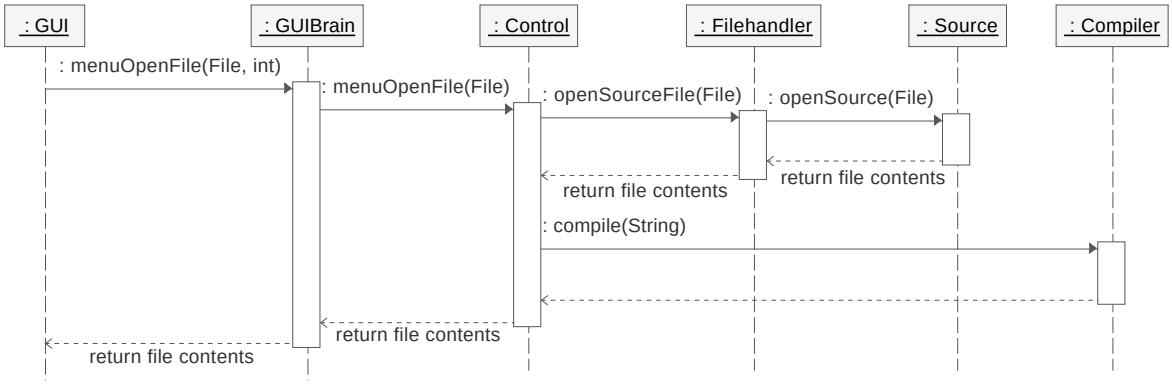
- Luodaan FileHandler- tai AssariUI-luokasta.
- Käytetään Control-luokasta.
- Oletetun lähdekooditiedoston abstrahoiva dataluokka, joka palautetaan tiedostoa avatessa FileHandlerilta Control-luokalle.

SymbolicInterpreter

- Interpreter-luokan aliluokka.
- Luokkaa käytetään konekielen lukuesitysten symboliseen muotoon muuttamiseen.

SymbolTable

- Luodaan Compiler-luokasta.
- Käytetään Application- ja tallennetaan RandomAccessMemoryyn TTK91-sovelluksen lataamisen yhteydessä.
- Dataluokka, joka sisältää kääntämisvaiheessa määritellyn symbolitaulun.



Kuva 3: .k91-tiedoston avaaminen

TitoKone

- Luokka on simulaattorin käynnistysluokka.
- Luo luokan GUI.

Translator

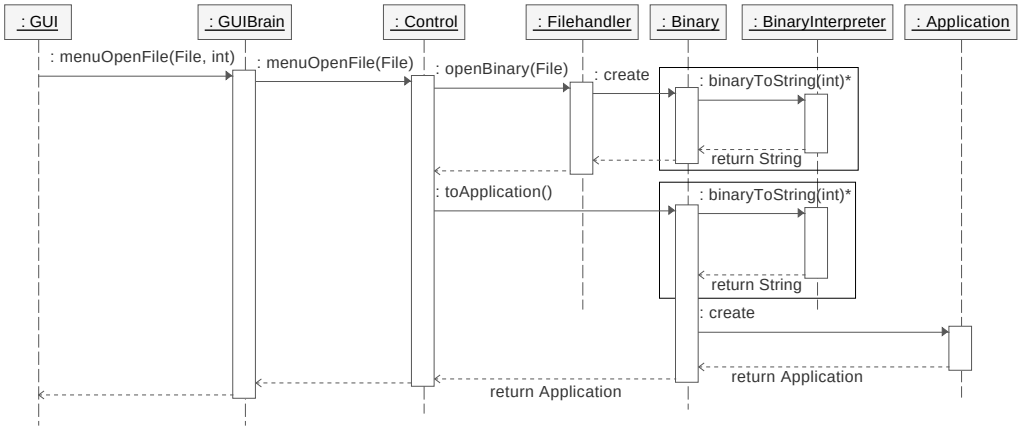
- Kielituen keskeisin luokka, jota käytetään Message-luokasta ja jonka asetuksia muutetaan GUIBrainista staattisesti.

4 Käyttötapaukset ja olioyhteistyö

Vaatimusmäärittelyssä määriteltiin erilaisia palveluja, joita ohjelma tarjoaa käyttäjälle. Näiden taustalla oli ajatus asioista, joita käyttäjä haluaisi tehdä, mutta näitä ei toiston välttämiseksi erikseen siellä käsitelty. Tässä dokumentissa käyttötapauksia katsotaan hieman toisesta näkökulmasta, jossa keskitytään käyttötapauksen toteutumiseen järjestelmässä ja siihen, miten eri luokat toimivat yhdessä tämän saavuttamiseksi. Käyttötapausten kuvauksessa käytetään oletuksena englanninkielisen käyttöliittymän termistöä, eikä niissä mainita erikseen komennoille varattuja pikanäppäimiä tai mahdollisia käyttöliittymän lisänappuloita. Joissakin kohdissa mainittu käyttäjän “jatka” -syöte vastaa rivinvaihtonäppäimen painallusta, Continue-napin klikkausta hiirellä, File -> Continue -valintaa tai Continue without stopping -napin tai valinnan klikkausta. eAssari-käyttöliittymällä tehtävät käyttötapaukset on kuvailtu erikseen, ja oletuksena puhutaan simulaattorikäyttöliittymästä.

4.1 Uuden TTK-91-ohjelman valinta

Käyttötapaus: Käyttäjä haluaa valita joko binäärimuotoisen .b91-ohjelman tai symbolisella konekielellä kirjoitetun .k91-ohjelman käsiteltäväksi.



Kuva 4: .b91-tiedoston avaaminen

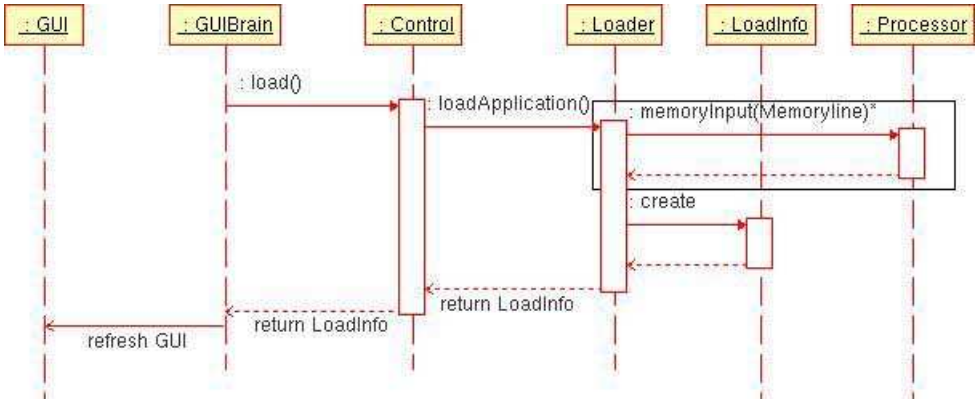
Käyttäjän toiminta: Käyttäjä valitsee valikosta File -> Open. Tämä avaa tiedostonvalinta-dialogin, jossa käyttäjä voi hiiren tai näppäimistön avulla selata hakemistonäkymää ja valita avattavan tiedoston mistä tahansa hakemistosta. Vaihtoehtoisesti hän voi kirjoittaa tiedoston polun suoraan dialogin tekstikenttään. Dialogissa näytetään vain b91- ja k91-päätteiset tiedostot, ja näistäkin vain yksi tyyppi kerrallaan käyttäjän valinnan mukaan.

Tulosteet: Mikäli kyseessä on .k91-ohjelma, näytetään ohjelman symbolinen konekielikäskylistaus. Mikäli kyseessä on .b91-muotoinen ohjelma, ohjelma ladataan ja käyttäjälle näytetään koneen sisäinen tila latauksen jälkeen. Ohjelman tyyppi päätellään sen tiedostopäätteestä.

Luokkien yhteistyö: GUI välittää selvittämänsä tiedostonimen (File-objekti) GUIBrain-luokalle, joka välittää tiedon Control-luokan openSource(File)- tai openBinary(File) kautta FileHandler-luokan tiedostonavauspalvelulle loadSource(File) tai loadBinary(File). Control-luokka tallentaa .k91-tiedoston polun itselleen voidakseen päätellä siitä jälkeenpäin käännetyn tiedoston tallennustiedoston.

Mikäli kyseessä on .k91-muotoinen tiedosto, FileHandler-luokka palauttaa Source-luokan ilmentymän, jonka Control tallettaa itselleen ja palauttaa sen merkkijonosisällön GUIBrainille. GUIBrain puolestaan valmistelee merkkijonon GUI:lle sopivan muotoiseksi, jotta tämä voidaan näyttää lähdekoodina. Lisäksi GUIBrain kertoo GUI:lle, että komento File -> Compile on nyt mahdollinen ja että File -> Run ei ole, mikäli se oli tätä toimintoa ennenkään. Mikäli taas kyseessä on .b91-muotoinen tiedosto, FileHandler-luokka palauttaa Binary-luokan ilmentymän. Control pyytää Binary-luokkaa tulkitsemaan itsestään (BinaryInterpreter-luokan avulla) Application-luokan ilmentymän, jonka se tallettaa itselleen.

Tämän jälkeen, mikäli kyseessä siis oli .b91-muotoinen tiedosto, GUIBrain kutsuu Control-luokan latauspalvelua. Control tarkistaa, oliko sovelluksella omat asetuksensa sille, mihin tiedostoihin sovelluksen aikaiset levyoperaatiot tehdään. (Tiedostojen vaihtamisen käyttötapaukset on kuvailtu luvuissa 4.8 ja 4.9.) Seuraavaksi se välittää tallentamansa Application-ilmentymän Loaderille, joka tulkitsee sen sisäl-



Kuva 5: K.91- ja B.91 tiedostojen lataaminen

lön ja kutsuu Processorin metodeja, joiden avulla sovellus ladataan SymbolTable- ja RandomAccessMemory-luokkiin ja Processorin tila päivitetään. Tämän jälkeen Loader pyytää Processorilta sen oleelliset tilatiedot, kokoaa niistä LoadInfo-ilmentymän ja palauttaa sen Controlille. Tämän jälkeen se palauttaa LoadInfon GUIBrainille.

GUIBrain tulkitsee LoadInfon ja pyytää GUI-luokkaa päivittämään itseään tarvittavilta osin. Lopuksi GUIBrain kertoo GUI:lle komennon File -> Run olevan mahdollinen.

Mahdolliset virheet: Binääritiedoston tulkkaaminen voi epäonnistua joko siksi, että tiedosto ei ole olemassa, käyttäjä ei sitä voi lukea tai tiedosto sisältää syntaksiltaan virheellistä .b91-tietoa. FileHandler luokka huomaa tiedoston poissaolon tai lukemisongelmat, ja Binary-luokka huomaa syntaksivirheet. Kumpikin heittää tällöin poikkeuksen ylävirtaan, jonka GUIBrain ottaa kiinni ja muokkaa GUI:ta varten virheilmoitukseksi. Tämä näytetään käyttäjälle omassa dialogissaan, minkä jälkeen GUI pyytää käyttäjää valitsemaan uuden tiedoston ja kiertokulku toistuu.

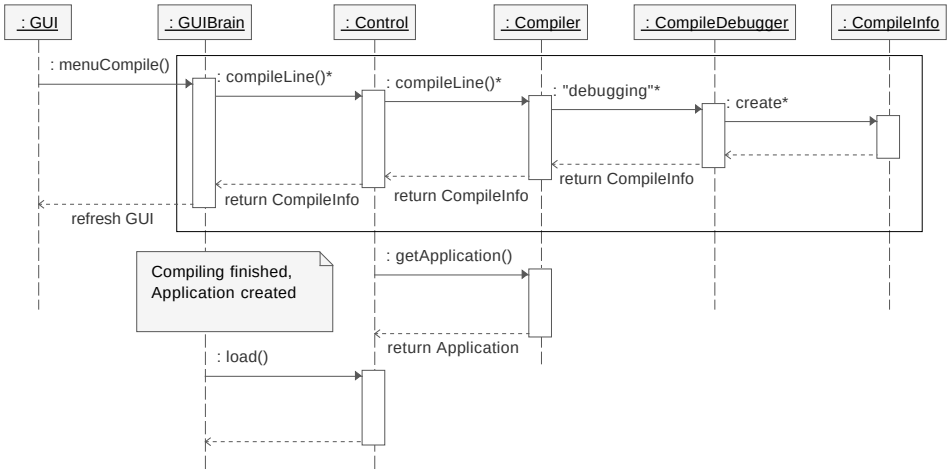
StdIn- ja StdOut-tiedostojen vaihtamiseen liittyy muita virheitä, joita on käsitelty tiedostojen uudelleenvalitsemista käsittelevissä luvuissa 4.8 ja 4.9.

4.2 TTK-91-ohjelman kääntäminen

Käyttötapaus: Käyttäjä haluaa kääntää aiemmin Open-komennolla avaamansa .k91-tiedoston binääriseksi konekielikoodiksi.

Käyttäjän toiminta: Käyttäjä valitsee valikosta File -> Compile. Tämä vaihtoehto on saatavilla vain silloin, kun ensin on valittu .k91-tiedosto, ks. edellä.

Tulosteet: Mikäli käyttäjä on valinnut kääntämisen kommentoitavaksi, niin hänelle näytetään sen edetessä kommentteja tapahtumista tyyliin "Symbolin ’paino’ arvo 80". Lopuksi käännetty ohjelma ladataan Titokoneen muistiin ja käyttäjälle näytetään koneen tila lataamisen jälkeen.



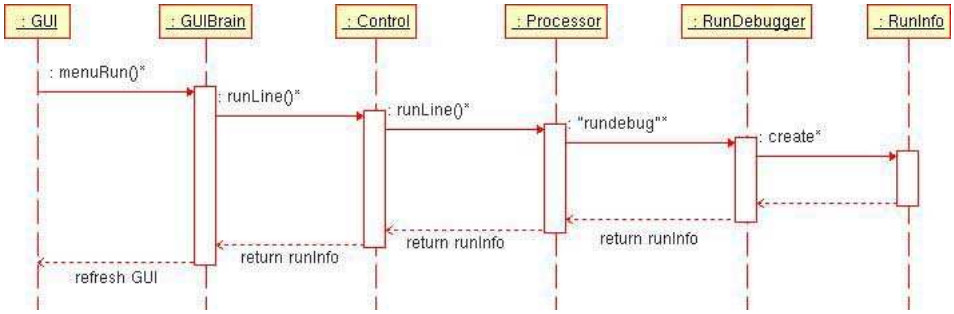
Kuva 6: TTK-91-ohjelman kääntäminen

Luokkien yhteistyö: GUI kertoo GUIBrainille kääntökomennosta. GUIBrain kertoo Control-luokalle uuden käännöksen alkavan, minkä tiedon Control välittää Compiler-luokalle. Tämän jälkeen GUIBrain siirtyy keskeytettävään tilaan, jossa käyttäjä voi pysäyttää toiminnon niinä hetkinä, joina häneltä odotetaan syötettä työskentelyn jatkamiseksi, tai mahdollisesti pienellä viiveellä silloin kun häneltä ei erityisesti odoteta jatkokomentoja. GUIBrain kutsuu toistuvasti Control-luokan palvelua yhden rivin kääntämiseksi. Control välittää käskyn Compiler-luokalle, joka tulkitsee rivin ja pyytää CompileDebuggerilta CompileInfo-ilmentymän, joka kertoo käännöksen etenemisestä. Ilmentymä palautetaan Compilerin kautta GUIBrainille, joka tulkitsee sen kommentointiasetuksista riippuvaan käyttäjälle näytettävään muotoon GUI:ta varten.

Tämän jälkeen, mikäli riveittäin kääntäminen on pois päältä, kääntökutsua toistetaan kunnes Control ei enää palauta CompileInfoja. Mikäli riveittäin suoritus on päällä, toistoa ennen odotetaan käyttäjältä jatkamaan ohjaavaa syötettä tai komentoa keskeyttää toiminto. Käännöksen keskeytys ei vaadi muuta kuin GUI-luokan tilan palauttamisen käännöstä edeltäväksi. Kun Control-luokka ei enää saa Compiler-luokalta uutta CompileInfoa, se pyytää Compileria kokoamaan Application-ilmentymän ja tallentaa sen itselleen ennen kuin välittää GUIBrainille CompileInfon palauttamisen sijaan saman tyhjäarvon. Lopuksi GUIBrain kutsuu Control-luokan latauspalvelua, joka on kuvailtu ohjelman valinnassa luvussa 4.1, ja pyytää GUI-luokkaa mahdollistamaan File -> Run -komennon.

Lisäksi GUIBrain kutsuu Control-luokan palvelua saveBinary(), jolla se tallentaa Application-luokan .b91-muotoisena tiedostona levylle työhakemistoon. Control-luokka luo uuden Binary-ilmentymän (Application parametrinä) ja pyytää sitä kääntämään itsensä merkkijonoksi, jonka se välittää arvelemansa tiedostonimen kanssa FileHandlerille tallennettavaksi. Tiedostonimi johdetaan aiemmin lähdekoodia avattaessa tallennetusta .k91-tiedostonimestä muuttamalla pääte .b91:ksi.

Mahdolliset virheet: Uutta riviä käännettäessä voidaan huomata symbolisen konekieli-



Kuva 7: Ohjelman ajaminen

komennon olevan virheellinen. Tällöin Compiler heittää poikkeuksen ylävirtaan, jonka GUIBrain ottaa kiinni ja näyttää virheilmoituksen GUI:n avulla käyttäjälle kommentointi-ikkunassa.

.b91-tiedoston tallentaminen ei välttämättä onnistu, mikäli käyttäjä ei voi kirjoittaa hakemistoon tai mikäli levy on täynnä. Tällöin GUIBrain pyytää GUI-luokan avulla käyttäjää valitsemaan tiedoston, johon .b91-tiedosto voidaan tallentaa. Käyttäjä voi myös painaa Cancel-nappia, jolloin .b91-tiedostoa ei tallenneta.

4.3 TTK-91-ohjelman ajaminen

Käyttötapaus: Käyttäjä haluaa suorittaa binäärimuotoisen ohjelman, joka on ensin joko suoraan luettu .b91-tiedostosta tai luettu .k91-tiedostosta ja käännetty.

Käyttäjän toiminta: Käyttäjä valitsee mahdollisesti ensin ajoasetukset, joita on kuvailtu myöhemmin, ja sen jälkeen valikosta File -> Run. Tämä komento on valittavissa vain kun käännetty (tai .b91-tiedostosta valittu) ohjelma on ladattu.

Tulosteet: Run-palvelu suorittaa ohjelman asetusten mukaisesti joko kaikki kerrallaan tai riveittäin ja kommenteilla, ilman tai animoiden ja kommenteilla. Riveittäin suoritettaessa käyttäjältä vaaditaan näppäimen tai hiiren napin painallusta suorituksen jatkamiseksi, kun kaikki kerrallaan suoritettaessa suoritus ajetaan “loppuun asti” kerrallaan. Tästä poikkeuksena ovat kuitenkin virhetilanteet tai tilanteet, jolloin TTK-91-ohjelma pyytää käyttäjältä syötettä.

Luokkien yhteistyö: GUI välittää komennon GUIBrainille, joka siirtyy keskeytettävään tilaan. Mikäli suoritus keskeytetään, GUIBrain odottaa käyttäjän “jatka”-syötettä, jonka jälkeen se käskee Control-luokkaa lataamaan ohjelman uudelleen. Keskeytettävään tilaan siirtymisen jälkeen GUIBrain alkaa kutsua toistuvasti Control-luokan compileLine()-palvelua. Control-luokka kutsuu vastaavasti Processor-ilmentymänsä compileLine()-palvelua. Processor hakee seuraavan komennon RandomAccessMemory-luokan edustamasta muistista, tulkitsee sen sekä muuttaa RandomAccessMemoryn tilaa ja omaansa komennon pyytämällä tavalla. Samalla Processor kertoo metodikutsuin RunDebuggerille, mitä tekee, ja RunDebugger koostaa tästä RunInfo-luokan ilmentymän, jonka Processor pyytää komentonsa suorituksen lopuksi Run-

Debuggerilta. Tämä palautetaan Control-luokan kautta GUIBrainille, joka tulkitsee siitä käyttäjälle näytettävät tilanmuutokset GUI:n esitettäväksi ja mahdollisesti välittää RunInfon vielä Animator-luokalle komennon animoinnin mahdollistamiseksi. Tilanmuutostiedot pysyvät samoina, mutta niistä näytetään käyttäjälle vain asetusten vaatimat tiedot. Esimerkiksi kun animointi ei ole päällä, siihen liittyviä tietoja ei tarvitse koota RunInfo-luokasta.

Mikäli vuorossa oleva komento yrittää lukea tietoa levyltä tai näppäimistöltä, Processor tarkistaa ensin syötepuskuristaan, onko siellä valmista luettavaa. Mikäli näin ei ole, Processor heittää poikkeuksen epäonnistuneesta luvusta. Control ottaa sen kiinni ja yrittää lukea Application-oliostaan tarvittavaa dataa. Mikäli dataa löytyy, Control lisää sen Processorin lukupuskuriin metodeilla keyboardInput(int) tai stdinInput(int). Mikäli ei, vastaavanlainen poikkeus heitetään ylöspäin, jolloin GUIBrain ottaa sen kiinni. Poikkeuksen tyypistä riippuen GUIBrain joko näyttää virheilmoituksen (StdIn-dataa ei enää ole) tai pyytää käyttäjältä automaattisesti tarkastettavaa syötettä GUI-luokan kautta (Kbd-datan puutoksen korjaamiseksi). Kun syötettä on saatu, GUIBrain kutsuu GUI:n herättämänä Control-luokan metodia keyboardInput(int). Tällöin Control-luokka kutsuu Processorin metodia keyboardInput(int) datan liittämiseksi sen lukupuskuriin ja käskee sitä uudelleenajamaan käskyn komennolla runLine(). Processor tietää edellisen komennon keskeytyneen, joten se ajaa sen uudelleen.

Kun Control-luokalta ei enää saada välitettynä palautusarvona uusia RunInfoja eli TTK-91-ohjelman suoritus on valmis, GUIBrain siirtyy keskeytettävästä tilasta odottamaan käyttäjältä jatkamiskomentoa asetuksista riippumatta. Kun tämä jatkamiskomento on annettu, GUIBrain nollaa muistin (kuvattu myöhemmin luvussa 4.5) ja lataa TTK-91-sovelluksen uudelleen (lataus kuvattu edellä luvun 4.1 lopussa).

Mahdolliset virheet: Suoritusvuorossa oleva TTK-91-käsky voi olla virheellinen parametreiltään tai komentokoodiltaan. Processor-ilmentymä heittää asiantilan todetessaan poikkeuksen, jonka GUIBrain ottaa kiinni ja tulkitsee käyttäjälle näytettäväksi virheilmoitukseksi GUI-luokalle.

Suoritusvuorossa oleva TTK-91-käsky voi yrittää kirjoittaa levylle, jonka seurauksena tuloste yritetään kirjoittaa myös fyysiselle levylle. Mikäli levy on täynnä, FileHandler heittää ylävirtaan I/O-poikkeuksen jonka seurauksena Control heittää oman kirjoituksen epäonnistumisesta kertoneen poikkeuksensa, joka kiertää Processorin kautta kutsupinoa alaspäin GUIBrainille. Tämä näyttää GUI:n avulla käyttäjälle virheilmoituksen ja keskeyttää suorituksen kuten jos käyttäjä olisi pyytänyt toiminnan keskeytystä riveittäin suorituksessa. Tämä on kuvailtu luvussa 4.4.

Levytiedostojen tarvittavat kirjoitus- ja lukuoikeudet tarkistetaan ohjelman käynnistyessä ja kun tiedostojen nimi vaihdetaan. Mikäli kuitenkin käyttäjä muuttaa oikeuksia kesken ohjelman ajon, FileHandler heittää poikkeuksen kuten levyn ollessa täynnä. Poikkeuksen käsittely tapahtuu samaan tapaan kuin yllä.

4.4 Toiminnon keskeyttäminen

Käyttötapaus: Käyttäjä seuraa riveittäin käännöstä tai riveittäin suoritusta ja haluaa keskeyttää toiminnon esimerkiksi koska suoritettava ohjelma on jäänyt ikuiseen silmukkaan.

Käyttäjän toiminta: Käyttäjä valitsee valikosta File -> Stop. Valinta on mahdollinen vain kun TTK-91-ohjelmaa ajetaan tai käännetään, eli jolloin GUIBrain on keskeytettävässä tilassa.

Tulosteet: Käännös tai ohjelman suoritus keskeytyy ja käyttäjälle näytetään tulostila siihen asti, kunnes hän käskee syötteellään ohjelmaa "jatkamaan" kuten rivi kerrallaan suorituksessa. Tällöin käännöksen keskeydyttyä käyttöliittymä palautetaan tilaan ennen käännöksen alkamista, tai ohjelman suorituksen keskeydyttyä käyttöliittymä palautetaan tilaan juuri ohjelman lataamisen jälkeen.

Luokkien yhteistyö: GUI käskee GUIBrainia keskeyttämään nykyisen toimintansa. GUIBrain jättää tästä itselleen maininnan sisäiseen muuttujaan, jonka se huomaa seuraavalle käännösriville tai seuraavaan suoritettavaan komentoon siirtyessään ja keskeyttää toimintansa, jättäen käyttöliittymänäkymän edellisen rivisuorituksen loppumisen jälkeiseen tilaan. Kun käyttäjä ilmoittaa aiemmin kuvaillulla "jatka"-syötteellä tarkastelleensa keskeytettyä tilaa tarpeekseen, GUIBrain palauttaa sisäisen tilansa joko .k91-tiedoston valinnan tai ohjelman latauksen jälkeiseksi keskeytetystä toiminnosta riippuen. Tämä toteutetaan .k91-tiedoston valinnan suhteen palauttamalla talletettu source-merkkijono, ja suorituksen keskeydyttyä siten, että GUIBrain pyytää Controlia lataamaan suoritettavan sovelluksen uudelleen.

Mahdolliset virheet: Tähän toimintoon ei liity käyttäjästä riippuvia virheitä.

4.5 Muistin tyhjentäminen

Käyttötapaus: Käyttäjä haluaa nollata muistin ja rekisterit. Esimerkiksi kun muistiin on aluksi ladattu 20 komentoa pitkä ohjelma ja sen jälkeen muistiin ladataan vain 10 komentoa pitkä ohjelma, edellisen ohjelman loppuosa jää muistiin. Sitä ei suoriteta eikä se vaikuta jälkimmäisen ohjelman suoritukseen, ellei jälkimmäisessä ohjelmassa sitä erikseen sallita esimerkiksi ohjelmointivirheen seurauksena, mutta sekalaiset jäännösluvut ympäri muistia voivat silti hämmentää käyttäjää. Tästä syystä hän voi päättää tyhjentää muistin ennen seuraavan ohjelman lataamista.

Käyttäjän toiminta: Käyttäjä valitsee valikosta File -> Erase memory.

Tulosteet: Jokaisen muistirivin ja rekisterien sisältö on tämän jälkeen 0. Lisäksi File -> Run -komento poistuu mahdollisesti ladatun ohjelman näin kadotessa käytöstä.

Luokkien yhteistyö: GUI kertoo komennosta GUIBrainille metodilla eraseMemory(). GUIBrain päivittää sisäistä tilaansa, kertoo tarvittaessa GUI:lle File -> Run -komennon poistuvan käytöstä ja tämän jälkeen kutsuu Control-luokan muistintyhjennys-

palvelua metodilla eraseMemory(). Control-luokka kutsuu vastaavasti Processorin muistintyhjennyspalvelua eraseMemory(), ja Processor päivittää oman tilansa sekä RandomAccessMemoryn ja SymbolTablen tilat nollatuiksi luomalla niistä uudet instanssit.

Mahdolliset virheet: Toimintoon ei liity käyttäjästä riippuvia virheitä.

4.6 Ohjelmasta poistuminen

Käyttötapaus: Käyttäjä haluaa poistua ohjelmasta.

Käyttäjän toiminta: Käyttäjä valitsee valikosta File -> Exit tai sulkee ohjelman pääikkunan.

Tulosteet: Ohjelma suljetaan.

Luokkien yhteistyö: GUI kutsuu System.exitiä.

Mahdolliset virheet: Toimintoon ei liity käyttäjästä riippuvia virheitä.

4.7 Kieliasetusten muuttaminen

Käyttötapaus: Käyttäjä haluaa vaihtaa käyttöliittymän kielen.

Käyttäjän toiminta: Käyttäjä valitsee valikosta Settings -> Choose Language ja avautuvasta alivalikosta joko haluamansa kielen järjestelmän mukana tulevien kielten joukosta tai vaihtoehdon Other..., josta aukeaa tiedostonvalintadialogi käyttäjän oman käännöstiedoston etsimiseksi. Mikäli käyttäjä huomaa kesken oman käännöstiedostonsa etsimisen ettei löydäkään sellaista, hän voi peruuttaa toimenpiteen, jolloin käyttöliittymän kieli pysyy entisellään.

Tulosteet: Käyttöliittymän kielivalinta päivitetään uusien viestien osalta, ja sen näkyvät käännettävät osat, kuten valikot, päivittyvät uuden kielen mukaisiksi. Kuitenkin esimerkiksi vanhoja kommentteja kommentti-kentässä ei päivitetä, vaan ne pysyvät samankielisinä kuin ne olivat kirjoitushetkellään.

Luokkien yhteistyö: GUI kertoo valinnasta GUIBrainille. Käytettävä metodikutsu riippuu siitä, valittiinko uusi kieli vaihtoehtojen joukosta vai oman käännöstiedoston kautta. Mikäli kyseessä on oman käännöstiedoston valinta, GUIBrain kutsuu tämän perusteella Control-luokan vastaavaa metodia, ja Control kutsuu FileHandleria avaamaan käännöstiedoston ja luomaan siitä resurssiluokan. Tämän jälkeen GUIBrain käskee staattista Translator-luokkaa päivittämään kielensä ja välittää sille tarvittaessa resurssiluokan. Mikäli kaikki on toiminut odotetusti, GUIBrain käskee Settings-luokkaa tallentamaan uuden asetuksen. Tämän jälkeen se lähettää Controlin kautta FileHandlerille Settings-luokan merkkijonomuodon (Settings-luokalla on tätä varten metodi toString()), jolloin FileHandler tallentaa luokan levylle. Lopuksi GUIBrain käskee GUI-luokkaa päivittämään ulkoasunsa uuden kielen mukaiseksi.

Mahdolliset virheet: Käyttäjän valitsema oma käännöstiedosto ei välttämättä ole tulkittavissa resurssitiedostoksi, jolloin FileHandler heittää poikkeuksen ylävirtaan GUIBrainille, joka näyttää käyttäjälle virheilmoituksen ja antaa tälle mahdollisuuden valita uusi tiedosto.

Ohjelman osana toimivien käännöstiedostojen katoamisen ei katsota olevan käyttäjästä riippuva virhetilanne, mutta mikäli näin tapahtuu, käyttäjälle näytetään virheilmoitus.

Asetusten tallentaminen tiedostoon ei välttämättä onnistu, jolloin käyttäjälle näytetään varoitus kommenttikentässä.

4.8 StdIn-tiedoston vaihtaminen

Käyttötapaus: Käyttäjä haluaa vaihtaa oletustiedoston, jota Titokone voi lukea IN-käskyn parametrillä =STDIN.

Käyttäjän toiminta: Käyttäjä valitsee valikosta Settings -> Set default stdin. Hän voi tämän jälkeen valita avautuvasta tiedostonvalintadialogista haluamansa tiedoston mistä tahansa hakemistosta, tai kirjoittaa sen polun suoraan tekstikenttään. Käyttäjä voi myös peruuttaa toiminnon painamalla Cancel.

Tulosteet: Tämän jälkeen tapahtuvat konekäskyllä IN Ri, =STDIN tehtävät luvut haetaan valitusta tiedostosta, ellei ajettava TTK-91-sovellus määritä omia vastaavia tiedostojaan. Oletustiedosto on “stdin” työhakemistossa. Tiedoston sisältö luetaan kokonaisuudessaan talteen kun tiedosto on valittu (oletustiedosto luetaan Titokoneen käynnistyessä).

Luokkien yhteistyö: GUI kutsuu GUIBrainin metodia menuSetStdIn(File) ja välittää tiedostopolun parametrinä. GUIBrain tallettaa asetukset itselleen kutsumalla Settings-luokan saveSetting(String, Object)-metodia ja kutsuu Controlin setStdIn(File)-metodia. Control pyytää FileHandleria avaamaan tiedoston kutsumalla loadStdIn(File)-metodia, jolloin sen sisältö palautetaan merkkijonobufferina. Tämän jälkeen Control kutsuu Application-luokan staattista palvelua checkInput(String) varmistaakseen, että syöte on oikeanmuotoista. (Mikäli näin ei ole, käyttäjälle näytetään varoitus.) Tieto haetaan kuitenkin FileHandlerista uudelleen joka latauksen yhteydessä, ja tällöin myös voidaan näyttää samainen varoitus.

Ladattaessa, mikäli käännösaikana Applicationille on asetettu oma StdIn-tiedostonsa, Control lataa Applicationin StdIn-tiedoston mukaiset tiedot setStdIn()-metodilla sen puskuriin ennen Applicationin siirtämistä Loaderille. Loader ei puutu StdIn- ja Kbd-tietoihin. Mikäli StdIn-tiedostoa ei ole asetettu Applicationissa, Control käyttää käyttäjän valitsemaa oletustiedostoa, tai tämänkin puuttuessa “stdin”-tiedostoa nykyisessä hakemistossa. Käytännössä Control tallettaa yhteen File-kenttään StdIn-tiedoston, jota käyttää mikäli Application ei sitä määritä. Järjestelmäoletus “stdin” saadaan GUIBrainilta (tai AssariUIlta).

Lopuksi GUIBrain tallentaa asetukset tiedostoon kuten luvussa 4.7, Kieliasetusten muuttaminen, on kuvattu.

Mahdolliset virheet: Käyttäjällä ei välttämättä ole lukuoikeuksia tiedostoon tai tiedostoa ei ole olemassa. Tällöin käyttäjää pyydetään valitsemaan uusi tiedosto tai peruuttamaan toimenpide. Mikäli Titokoneen käynnistyessä luettu oletustiedosto sisältää jotakin muuta kuin sopivalla tavalla erotettuja lukuja, kommenttikentässä näytetään varoitus, mutta käyttäjän ei tarvitse valita uutta tiedostoa. Mikäli käyttäjä yrittää kuitenkin myöhemmin lukea IN Ri, =STDIN -konekäskyllä tiedostosta jota ei voida lukea, tapahtuu virhe ja suoritus keskeytyy.

Mikäli tiedosto sisältää muuta kuin sopivalla tavalla erotettuja lukuja, käyttäjälle näytetään välittömästi virheilmoitus ja häntä pyydetään valitsemaan uusi tiedosto. Mikäli Titokoneen käynnistyessä luettu oletustiedosto sisältää jotakin muuta kuin sopivalla tavalla erotettuja lukuja, kommenttikentässä näytetään varoitus. Kuten edellä, käyttäjä ei tällöin voi lukea tiedostoa ilman virheen tapahtumista.

Ongelmat asetusten kirjoittamisessa tiedostoon on käsitelty luvussa 4.7.

4.9 StdOut-tiedoston vaihtaminen

Käyttötapaus: Käyttäjä haluaa vaihtaa oletustiedoston, johon Titokone voi kirjoittaa OUT-käskyn parametrillä =STDOUT.

Käyttäjän toiminta: Käyttäjä valitsee valikosta Settings -> Set default stdout. Hän voi tämän jälkeen valita avautuvasta tiedostonvalintadialogista hakemiston, johon haluaa tallentaa, ja joko valita olemassaolevan tiedoston tai kirjoittaa uuden tiedoston nimen tekstikenttään. Myös tiedostonimi polkuineen voidaan kirjoittaa kenttään suoraan. Mikäli käyttäjän valitsema tiedosto on jo olemassa, käyttäjältä kysytään, haluaako hän ylikirjoittaa olemassaolevan tiedoston vai liittää uudet tiedot tiedoston perään. Käyttäjä voi peruuttaa StdOut-tiedoston vaihtamisen painamalla Cancel.

Tulosteet: Mikäli käyttäjä on valinnut olemassaolevan tiedoston ylikirjoittamisen, tiedosto tyhjennetään nyt. Tämän jälkeen tapahtuvat konekäskyllä OUT Ri, =STDOUT tehtävät kirjoitukset liitetään valitun tiedoston loppuun. Oletustiedosto on “stdout” työhakemistossa, ja kirjoitettavat tiedot liitetään tämän mahdollisesti jo olemassaolevan tiedoston perään.

Luokkien yhteistyö: GUI kutsuu GUIBrainin menuSetStdOut(File)-metodia. GUIBrain tarkistaa File-oliosta, oliko tiedosto jo olemassa, ja pyytää tarvittaessa GUI-luokkaa kysymään käyttäjältä, haluaako tämä liittää kirjoitettavat tiedot olemassaolevan tiedoston jatkeeksi vai ylikirjoittaa sen. Tämän jälkeen se joko uudelleenluo tiedoston tai välittää sellaisenaan sitä edustavan File-olion parametrinä Control-luokan setDefaultStdOut(File)-metodille, jolloin Control pyytää FileHandleriä yrittämään tiedoston avaamista siten, että kirjoitettavat tiedot liitetään olemassaolevan (mahdollisesti nyt tyhjätyn) tiedoston jatkeeksi metodilla testAccess(File,int).

Koska Application voi määrittää oman StdOut-tiedostonsa, tiedosto avataan uudelleen aina latauksen yhteydessä, samoin ylikirjoitusasetuksin kuin viimeksi. Mikäli Application määrittää oman tiedostonsa, oletuksena on ylikirjoitus. Control tarkistaa nämä määritykset ja muuttaa tilaansa vastaavasti ennen Applicationin välittämistä Loaderille.

Lopuksi GUIBrain tallentaa asetukset tiedostoon kuten luvussa 4.7, Kieliasetusten muuttaminen, on kuvattu.

Mahdolliset virheet: Käyttäjällä ei välttämättä ole oikeuksia kirjoittaa tiedostoon, jolloin näytetään virheilmoitus ja käyttäjää pyydetään valitsemaan uusi tiedosto. Mikäli oletustiedoston avaaminen kirjoitusta (liittämistä) varten epäonnistuu Titokoneen käynnistyessä, kommenttikentässä näytetään tästä varoitus, mutta käyttäjän ei tarvitse valita uutta tiedostoa. Hänen tulee kuitenkin vaihtaa tiedostoa mikäli hän aikoo ajaa STDOUTiin kirjoittavia ohjelmia, sillä muuten kirjoittavan komennon ajamisessa tapahtuu virhe.

Käyttäjän valitsema tiedostopolku voi myös olla tavalla tai toisella virheellinen, esimerkiksi mikäli se johtaa hakemistoon jota ei ole olemassa. Tässä tapauksessa käyttäjää pyydetään valitsemaan uusi tiedosto.

Ongelmat asetusten kirjoittamisessa tiedostoon on käsitelty luvussa 4.7.

4.10 Muistin koon vaihtaminen

Käyttötapaus: Käyttäjä haluaa vaihtaa Titokoneen keskusmuistin koon.

Käyttäjän toiminta: Käyttäjä valitsee valikosta Settings -> Set Memory Size ja valitsee alivalikosta saatavilla olevista vaihtoehdoista (512, 1024, 2048, 4096, 8192, 16384, 32768 tai 65536 sanaa l. datariviä) uuden muistin koon. Käyttäjältä kysytään varmistusta kuten luvussa 4.5, Muistin tyhjentäminen. Mikäli käyttäjä valitsee muistin kooksi 65536 (2^{16}), ylintä muistiavaruuden puoliskoa ei voi osoittaa suoraan koodista (suurin sallittu vakion arvo on $2^{15} - 1$). 32-bittiset rekisterit mahdollistavat kuitenkin mutkan kautta koko muistialueen (suuremmankin) osoittamisen.

Tulosteet: Mahdollinen käännös tai ohjelman ajo keskeytetään, muistin koko muutetaan ja sen sisältö tyhjennetään. Mikäli käyttäjä valitsi muistialueen kooksi suurimman, 66536, häntä varoitetaan edellä mainitusta suoran osoituksen rajoituksesta käyttöliittymän kommenttikentässä.

Luokkien yhteistyö: GUI kutsuu GUIBrainin metodia menuSetMemorySize(int) ja välittää parametrinä uuden muistin koon. GUIBrain kutsuu vastaavasti Control-luokan metodia setMemorySize(int), ja Control luo itselleen uuden Processor-luokan ilmentymän jonka muistin koko on oikea. Processor luo itselleen tämän mukaisesti RandomAccessMemory-luokan, jonka koko on annetunlainen, mikä samalla täyttää muistin MemoryLine-luokan nollasisältöä vastaavilla ilmentymillä. Lopuksi GUIBrain tallentaa asetukset tiedostoon kuten luvussa 4.7, Kieliasetusten muuttaminen, on kuvattu.

Mahdolliset virheet: Toimintoon ei liity käyttäjästä riippuvia virheitä aiemmin kuvatun asetustiedostoon kirjoittamisen epäonnistumisen lisäksi.

4.11 Käännösaikaisten asetusten muuttaminen

Käyttötapaus: Käyttäjä haluaa valita, käännetäänkö ohjelma rivi kerrallaan vai heti loppuun asti ja kommentoidaanko käännöstä sen edetessä. Näitä kahta vaihtoehtoa voidaan yhdistellä eri tavoin.

Käyttäjän toiminta: Käyttäjä valitsee valikosta Settings -> Set Compiler Options. Hänen eteensä avautuu ikkuna, josta hän voi valita kummankin asetuksen päälle tai pois päältä. Lopuksi hän voi klikata joko Apply, joka tallentaa muutetut asetukset sulkematta ikkunaa, tai Close, joka sulkee valintaikkunan tallentamatta asetuksia. Apply-nappi ei ole käytössä ennen kuin asetuksia on muutettu.

Tulosteet: Kun asetukset on talletettu, käännös suoritetaan jatkossa niiden mukaisesti. Mikäli käännöstä tehtiin riveittäin juuri silloin kuin asetuksista muutettiin käännös suoraan loppuun eteneväksi, käännös etenee pysähtymättä loppuun asti heti käyttäjän ilmoitettua syötteellä haluavansa käännöksen jatkuvan.

Luokkien yhteistyö: GUI kutsuu GUIBrainin metodia menuSetCompileOptions(boolean, boolean) ja välittää tiedon siitä, mitkä asetukset valittiin. GUIBrain tallentaa uuden asetuksen Settings-luokkaan ja tallettaa lopulta asetukset tiedostoon kuten luvussa 4.7, Kieliasetusten muuttaminen, on kuvattu.

Mahdolliset virheet: Toimintoon ei liity käyttäjästä riippuvia virheitä aiemmin kuvatun asetustiedostoon kirjoittamisen epäonnistumisen lisäksi.

4.12 Ajoaikaisten asetusten muuttaminen

Käyttötapaus: Käyttäjä haluaa valita, ajetaanko ohjelma rivi kerrallaan vai loppuun asti, kommentoidaanko ajoa sen tapahtuessa ja animoidaanko CPU:n tapahtumia tarkemmin erillisessä animointi-ikkunassa. Kahta ensimmäistä vaihtoehtoa voidaan yhdistellä eri tavoin, mutta animointi tapahtuu aina kommentoiden ja rivi kerrallaan.

Käyttäjän toiminta: Käyttäjä valitsee valikosta Settings -> Set Running Options. Hänen eteensä avautuu ikkuna, josta hän voi valita kunkin kolmesta asetuksesta päälle tai pois päältä. Kun käyttäjä valitsee animoinnin päälle, riveittäin ajo ja kommentointi valitaan automaattisesti, ja kun jompikumpi valitaan pois, myös animointivalinta poistuu. Lopuksi hän voi klikata joko Apply, joka tallentaa muutetut asetukset sulkematta ikkunaa, tai Close, joka sulkee valintaikkunan tallentamatta asetuksia. Apply-nappi ei ole käytössä ennen kuin asetuksia on muutettu.

Tulosteet: Kun asetukset on talletettu, ohjelmien ajaminen tehdään jatkossa niiden mukaisesti. Mikäli koodia suoritettiin riveittäin juuri silloin kuin asetuksista muutettiin suoritus suoraan loppuun eteneväksi, suoritus etenee pysähtymättä ohjelman loppuun heti käyttäjän ilmoitettua syötteellä haluavansa käännöksen jatkuvan.

Luokkien yhteistyö: GUI kutsuu tarpeen mukaan GUIBrainin metodia menuSetRunningOptions(int, boolean) ja välittää tiedon siitä, mitä asetuksia valittiin/muutettiin. GUIBrain tallentaa uuden asetuksen Settings-luokkaan ja tallettaa lopulta asetukset tiedostoon kuten luvussa 4.7, Kieliasetusten muuttaminen, on kuvattu.

Mahdolliset virheet: Toimintoon ei liity käyttäjästä riippuvia virheitä aiemmin kuvatun asetustiedostoon kirjoittamisen epäonnistumisen lisäksi.

4.13 Ohjelman tietojen tarkastelu

Käyttötapaus: Käyttäjä on inspiroitunut ohjelman käytöstä ja haluaa tietää kaiken sen tekijöistä ja saatavuudesta.

Käyttäjän toiminta: Käyttäjä valitsee valikosta Help -> About.

Tulosteet: Käyttäjälle avataan dialogi-ikkuna, jossa ainakin näytetään ohjelman versiotiedot, kerrotaan lyhyesti sen taustasta ja annetaan linkki sivulle, josta asiakas hallinnoi ohjelman levitystä, siis ohjelman kotisivulle.

Luokkien yhteistyö: GUI näyttää sisäisesti tallennetun About-tekstin.

Mahdolliset virheet: Toimintoon ei liity käyttäjästä riippuvia virheitä.

4.14 Ohjelman käyttöohjeen tarkastelu

Käyttötapaus: Käyttäjä haluaa ottaa sovelluksesta kaiken irti, ja kaipaa hienouksien esittelyyn apua laajemmasta käyttöohjeesta kuin mitä ohjelman käytön ohessa näyttämät työkaluvihjeet tarjoavat.

Käyttäjän toiminta: Käyttäjä valitsee valikosta Help -> Manual.

Tulosteet: Käyttäjälle avataan ikkuna, jossa hän voi selata käyttöohjetta. Käyttöohjeessa mainitaan myös, missä osoitteessa sitä voi selata myös webissä.

Luokkien yhteistyö: GUI näyttää kovakoodatusta tiedostopolustaan löytyvän HTML-tiedoston sisällön erillisessä ikkunassa.

Mahdolliset virheet: Käyttöohjetiedoston avaamisen epäonnistuminen on vakava, mutta käyttäjästä riippumaton virhe, josta ilmoitetaan käyttäjälle virheilmoituksella.

5 Käyttöliittymä

Ohjelman rakenne on suunniteltu tukemaan kahta käyttöliittymää: simulaattoria ja eAssari-järjestelmää. Simulaattorin on tarkoitus kuvata konekielitasolla tapahtuvaa toimintaa prosessorilla. eAssari-käyttöliittymä on tarkoitettu konekielisten ohjelmien ajamiseen ja tilatietojen tulostamiseen siten, että opiskelijalle tehtäväksi annetun ohjelman toiminnan tarkastelu voidaan automatisoida ja tehtävä tarkistaa ohjelmallisesti. Käyttöliittymää eAssarille ei toteuteta, vaan ainoastaan käyttöliittymälle määritelty rajapinta ja testauksen mahdollistava käyttöliittymätynkä AssariUI.

Tässä luvussa esiteltävällä käyttöliittymällä tarkoitetaan vain ja ainoastaan ohjelman varsinaista käyttöliittymää, ei eAssariin toteutettavaa käyttöliittymää.

Tämän ohjelman graafinen käyttöliittymä toteutetaan Java Swing -ikkunointiympäristöllä. Se koostuu pääikkunasta, jonka ohjelman tilasta riippuvat kolme eri näkymää on tässä esitelty, sekä animointi-ikkunasta. Näkymät on selostettu niille ominaisten toimintojen suhteen yksityiskohtaisesti kolmessa seuraavassa aliluvussa, ja animointi-ikkuna hieman avoimemmin. Tietyt käyttöliittymän toiminnot ovat kuitenkin mahdollisia jokaisessa niistä ja ne on selostettu yksityiskohtaisesti seuraavaksi.

Käyttäjä voi avata uuden tiedoston (binäärisen tai lähdekooditiedoston), mikä tehdään valitsemalla toiminto *Open* valikosta *File* tai painamalla työkalupalkin nappia *open*. Kummatkin avaavat näytölle tiedostonvalintadialogin, josta käyttäjä voi valita haluamansa tiedoston graafisesta hakemistonäkymästä tai kirjoittaa sen tekstikenttään. Graafisessa hakemistonäkymässä näytetään käyttäjän valinnan mukaan ainoastaan joko b91- tai k91-päätteisiä tiedostoja. Mikäli .b91-tiedosto, jota yritetään avata, ei sisälläkään oikeanmuotoista binääriohjelmaa, ilmoitetaan siitä erillisellä dialogilla ja pyydetään valitsemaan uusi tiedosto. Mikäli tiedoston pääte on jokin muu kuin k91 tai b91, ilmoitetaan myös siitä käyttäjälle erillisellä dialogilla. K91-päätteinen tiedostohan voidaan avata jokatapauksessa, sillä siinä olevat virheet huomataan vasta käännösvaiheessa. Onnistunut tiedostonavaus muuttaa näkymän #2:ksi (lähdekoodinäkymä). Mikäli avaamisessa tapahtuu jokin muu virhe, siitäkin ilmoitetaan erillisessä dialogissa.

Käyttäjä voi myös muuttaa Titokoneen keskusmuistin kokoa. Se tapahtuu valitsemalla haluttu muistikoko valikon *Options* alivalikosta *Set memory size*. Valinnan jälkeen ilmestyy dialogi, joka varoittaa käyttäjää siitä, että muistin sisältö nollautuu samalla ja kysyy varmistuksen toiminnolle. Myönteisessä tapauksessa muistin koko muuttuu valituksi ja näkymä muuttuu #1:ksi (alkunäkymä).

Käyttäjä voi muuttaa oletuksena käytettäviä stdin- ja stdout-tiedostoja valitsemalla valikon *Options* alivalikosta *Configure filesystem* joko *Set default stdout* tai *Set default stdin*. Siitä ilmestyy ruudulle tiedostonvalinta dialogi, josta valitaan haluttu tiedosto. Tässä tapauksessa graafisessa hakemistonäkymässä näytetään kaikki tiedostot ja käyttäjä voi myös valita minkä tahansa tiedoston. Avaamisessa saattaa ilmetä jotain virheitä, kuten että käyttäjällä ei ole luku- tai kirjoitusoikeutta tiedostoon. Näistä ilmoitetaan dialogilla ja pyydetään käyttäjää avaamaan uusi tiedosto. Lisäksi stdout-tiedoston valinnan jälkeen, mikäli tiedosto oli jo olemassa, käyttäjältä kysytään, haluaako hän liittää uudet tulosteet tiedoston loppuun vaiko ylikirjoittaa tiedoston. On huomattava, että mikäli TTK91-

Kuva 8: Set running options -dialogi

Kuva 9: Set compiling options -dialogi

ohjelman koodissa on ilmoitettu kyseiselle ohjelmalle oma stdin- tai stdout-tiedosto, tällä valinnalla ei ole merkitystä.

Kun käyttäjä valitsee valikosta *File* toiminnon *Erase memory*, tyhjentyy Titokoneen muisti ja käyttöliittymässä muutetaan näkymäksi #1 (alkunäkymä).

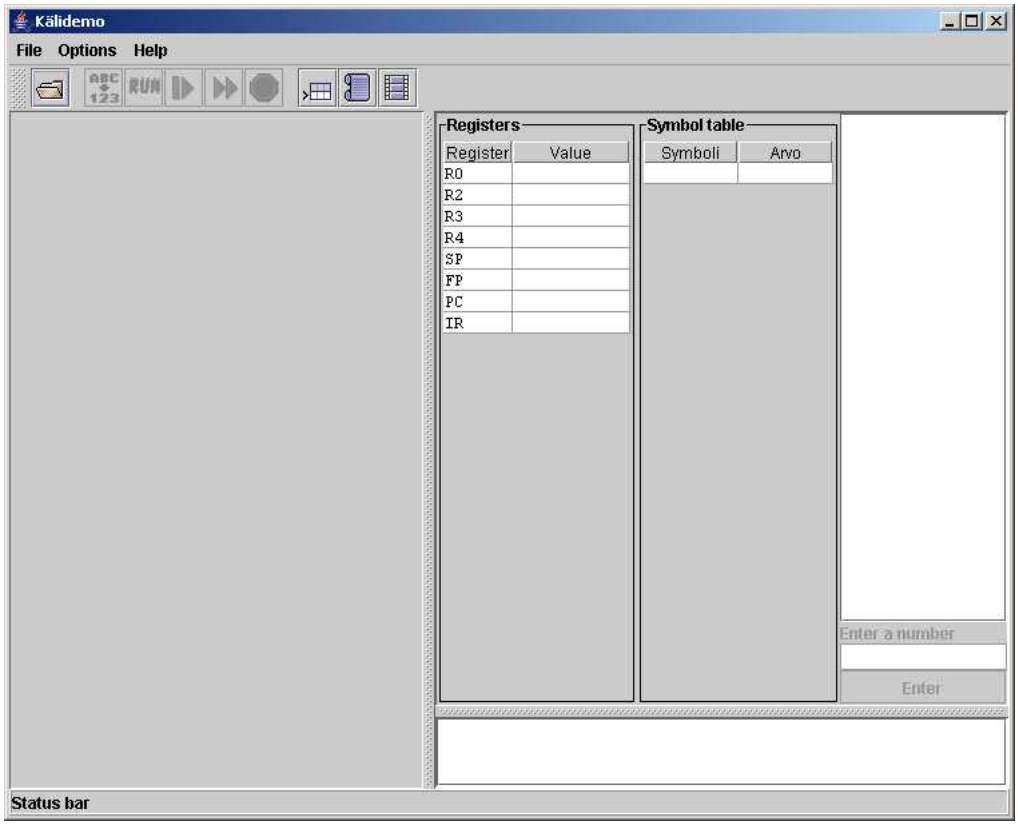
Käyttöliittymän kieli on vaihdettavissa valitsemalla haluttu kieli valikon *Options* alivalikosta *Choose language*. Siellä on listattuna valittavat kielet sekä lisäksi toiminto *Other...*, jonka valinnasta ilmestyy tiedostonvalintadialogi, Siitä käyttäjä voi valita jonkin kielitiedoston, joka on jossain muualla kuin oletushakemistossa, josta ohjelma osaa hakea kielet itse valikkoon.

Ajoaikaiset asetukset ovat muutettavissa dialogista, jonka käyttäjä voi avata valitsemalla toiminnon *Set running options* valikosta *File*. Dialogi näkyy kuvassa 8. Dialogissa voidaan valita ajetaanko ohjelmaa rivi kerrallaan tai vaihtoehtoisesti jollain tietyllä nopeudella; näytetäänkö kommentointi; ja näytetäänkö myös animointi. Näille kaikille on työkalupalkissa oma kytkimensä, jonka alaspainaminen vastaa sitä kuin kyseinen optio olisi valittu dialogista. Muutos näkyy myös dialogissa.

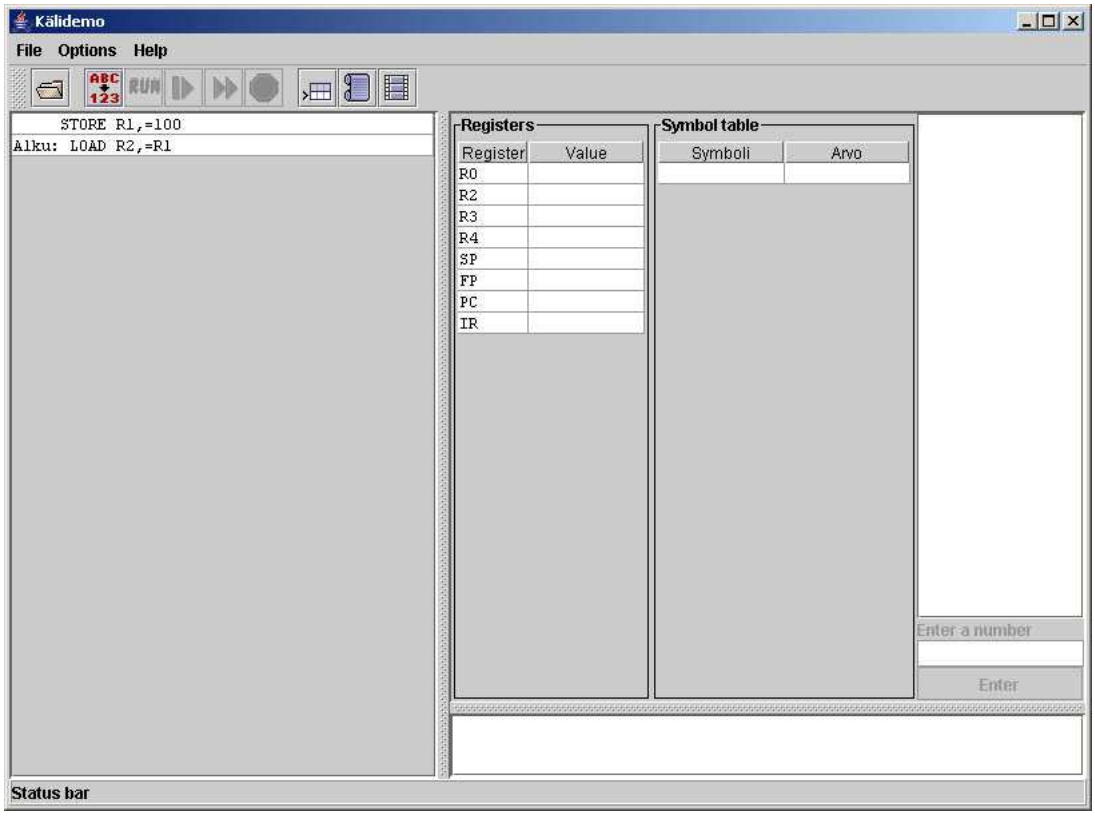
Käännösaikaiset asetukset muutetaan myös omasta dialogistaan. Dialogi näkyy kuvassa 9. Käyttäjä voi avata dialogin valitsemalla toiminnon *Set compiling options* valikosta *Options*. Dialogista hän voi valita, haluaako hän nähdä käännöksen kommentit ja keskeytetäänkö ohjelman suoritus kunkin rivin jälkeen.

5.1 Näkymä 1: Alkunäkymä

Tämä näkymä on aktiivinen silloin, kun käyttäjä ei ole avannut .k91-tiedostoa eikä Titokoneen muistiin ole ladattuna ohjelmaa. Tässä näkymässä käyttäjällä ei ole yllämainittujen yleistoimintojen lisäksi mitään erityisiä toimintoja, joten niitä ei tässä voida myöskään



Kuva 10: Näkymä 1: Alkunäkymä.



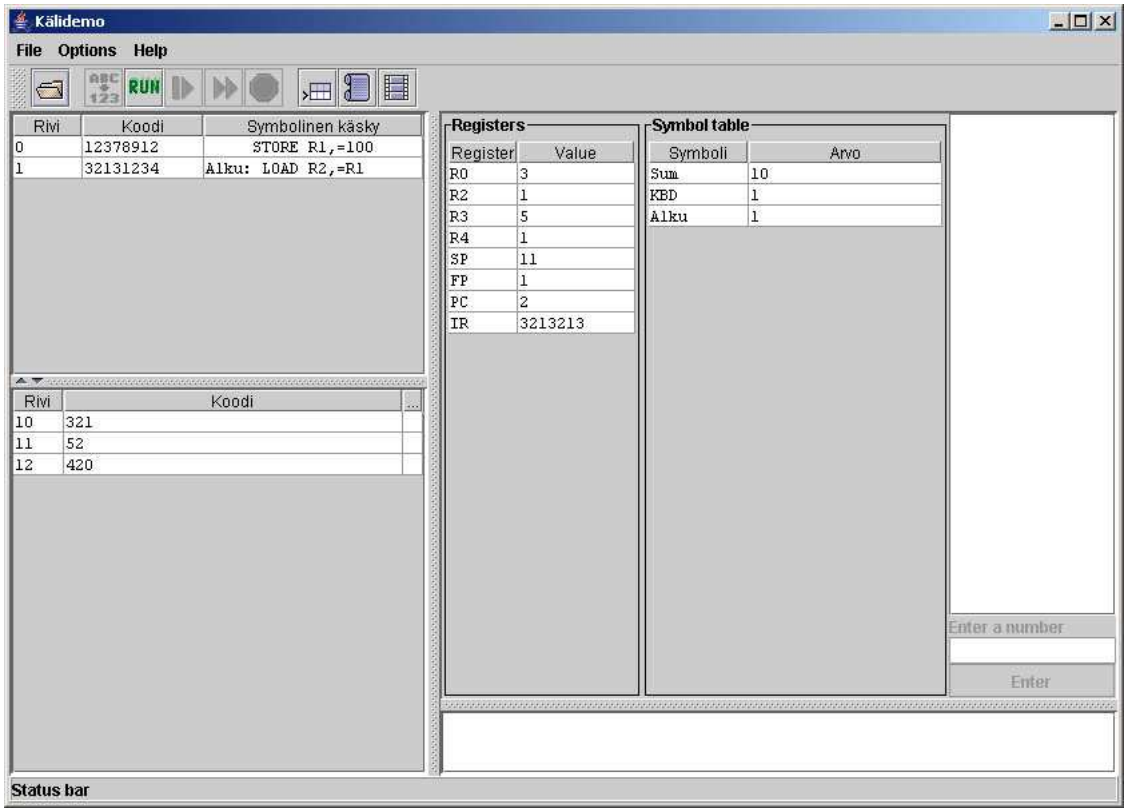
Kuva 11: Näkymä 2: Lähdekoodinäkymä.

luetella. Erityisesti kannattaa huomata, että vasemmalla puolella oleva paneeli on kyllä olemassa, vaikka se onkin tyhjä. Tämä siksi, että näkymät olisivat keskenään yhtenäisiä, sillä muissa näkymissä siihen tulee kyllä sisältöä. Valikossa *File* olevat toiminnot *Compile*, *Run*, *Stop* ja *Continue* sekä niitä vastaavat napit työkalupalkissa ovat aina poistettuina käytöstä tässä näkymässä.

5.2 Näkymä 2: Lähdekoodinäkymä

Tämä näkymä on aktiivinen, kun käyttäjä on avannut .k91-tiedoston, jota ei kuitenkaan ole vielä käännetty. Ikkunan vasemmassa reunassa näkyy avatun tiedoston sisältö. Käyttäjä voi kääntää ohjelman valitsemalla toiminnon *Compile* valikosta *File* tai painamalla vastaavaa nappia työkalupalkissa. Kääntämisen aikana suoritettava rivi on korostettuna. Mikäli käyttäjä on valinnut käännöksen kommentoitavaksi, sen aikana oikeassa alareunassa olevaan kenttään ilmestyy kommentteja sitä mukaa kuin käännös etenee. Jokaisessa kommentissa on mukana myös rivinumero, johon kyseinen kommentti viittaa.

Mikäli käyttäjä on valinnut, että kääntäminen keskeytetään aina, kun kommentti ilmestyy, niin operaatiota voidaan jatkaa painamalla rivinvaihtoa, valitsemalla toiminto *Continue* valikosta *File* tai painamalla vastavaa työkalupalkin nappia. Välilyönti ja hiiren klikkaus käyttöliittymän muissa osissa olivat myös suunnitelmissa, mutta Swing kaappaa välilyönnin ja hiiren klikkausten seurailu päätettiin suosiolla jättää toteuttamatta. Tällöin



Kuva 12: Näkymä 3: Sovellusnäkymä.

kääntämistä jatketaan heti seuraavalta riviltä. Tauon aikana korostettuna on rivi, joka oli suorituksessa ennen taukoa. Käyttäjä voi myös suorittaa koko käännöksen ilman taukoja kerralla loppuun painamalla työkalupalkissa olevaa *Fast forward* -nappia tai valitsemalla valikosta *File* toiminnon *Continue without stopping*. Käännöksen aikana vastaantulevat symbolimääritykset lisätään saman tien ikkunassa näkyvään symbolitauluun. Jos käännöksen aikana tulee vastaan virhe, siitä ilmoitetaan kommenttipaneelissa ja kääntäminen pysäytetään. Kääntämistä ei tämän jälkeen voi jatkaa vaan se on aloitettava alusta. Alkutilaan ei kuitenkaan siirrytä saman tien, vaan suoritus vain keskeytetään, jolloin korostetuksi jää rivi, jolla virhe oli. Tällöin käyttäjä voi tarkastella virheeseen johtanutta tilaa. Käännöksen alkutilaan siirrytään, kun käyttäjä antaa jatkamista pyytävän syötteen (joko toiminnolla *Continue* valikossa *File* tai vastaavalla napilla työkalupalkissa). Tämä alkutila vastaa tilannetta .k91-tiedoston avaamisen jälkeen.

Käyttäjä voi myös itse pysäyttää kääntämisen valitsemassa *File*-valikosta toiminnon *Stop* tai painamalla työkalupalkista vastaavaa nappia. Tällöin siirrytään alkutilaan, eli koodi-taulun sisältö muuttetaan siihen, mitä se oli ennen kääntämisen aloittamista ja symboli-taulusta tulee tyhjä, mikäli siihen oli jotain ehditty lisätä.

5.3 Näkymä 3: Sovellusnäkymä.

Tämä näkymä on aktiivinen, kun TTK-91-ohjelma on ladattu Titokoneen muistiavaruuteen joko .k91-muotoisen lähdekooditiedoston kääntämisen tai binäärimuotoisen .b91-tiedoston avaamisen yhteydessä. Ruudun vasemmanpuoleisessa osassa on allekkain kaksi taulukkoa, joista ylempää kutsutaan käskytauluksi ja alempaa datatauluksi. Kummatkin taulut tarjoavat näkymän Titokoneen muistiin, mutta ylemmässä näytetään vain muistin suoritettava osa, siis käskyosa, ja alemmassa näytetään loput muistista. Molemmissa tauluissa on kolme saraketta ja useita rivejä, missä yksi rivi vastaa yhtä muistiriviä. Ensimmäisessä sarakkeessa näytetään muistin rivinumero, toisella muistin sisältö kymmenkantaisena kokonaislukuna ja kolmannella tämän luvun symbolinen esitys. Datataulun näkymä eroaa kooditaulun vastaavasta siinä, että datataulu kolmas sarake on oletuksena pienennetty, koska data-alueen sisällön symboliset esitykset ovat merkityksettömiä. Saraketta ei kuitenkaan ole poistettu kokonaan, sillä käyttäjä saattaa haluta suurentaa sen, jos hän käyttää ohjelmassa itseään muuttavaa koodia, joka muuttaa myös data-alueen sisältöä konekielisten käskyjen koodeiksi.

Rekisteritaulussa näytetään rekisterien R0, R1, R2, R3, R4, PC, SP ja FP sisältö. Taulu päivitetään aina heti, kun rekisterin arvon muutoksesta saadaan tieto. Symbolitaulussa näytetään symbolisten vakioiden arvot, jotka on määritelty .k91-tiedostossa.

Ikkunan oikeassa yläkulmassa ovat kbd-input ja crt-output kentät. Ylempi ja suurempi on crt-output kenttä, johon kirjoitetaan Titokoneen crt-laitteelle kirjoittamia lukuja. Se siis tavallaan vastaa Titokoneen monitoria. Jokainen luku kirjoitetaan omalle rivilleen ja uusi luku lisätään edellisen alle. Kenttä on vieritettävä, mikäli lukuja on enemmän kuin siihen kerralla mahtuu. Sitä ei voi muokata suoraan. Kentän alla on pienempi kbd-input -kenttä, johon kirjoitetaan lukuarvo Titokoneen kysyessä sellaista. Kenttä on aktiivinen ainoastaan silloin, kun Titokone odottaa lukua. Odotuksesta ilmoitetaan myös kentän yläpuolelle ilmestyvällä “Enter a number” -tekstillä. Kirjoitettu luku syötetään koneelle joko painamalla näppäimistöstä rivinvaihtoa tai klikkaamalla kentän alapuolella olevaa *Enter*-nappia. Oikeassa alalaidassa olevaan kommenttipaneeliin ilmestyy tällöin ilmoitus luvun lukemisesta. Myös muut ajoaikaiset kommentit kirjoitetaan tähän kommenttipaneeliin.

Titokoneeseen ladattu ohjelma suoritetaan valitsemalla *File*-valikosta toiminto *Run* tai painamalla vastaavaa nappia työkalupalkista. Ohjelmaa suoritetaan rivi kerrallaan ja suorituksessa oleva rivi on korostettu. Mikäli valittuna on, että koodia suoritetaan rivi kerrallaan, jäädään jokaisen rivin jälkeen odottamaan käyttäjältä joko välilyönnin tai rivinvaihdon painallusta, *Continue*-toiminnon valintaa *File*-valikosta, vastaavan napin painamista työkalupalkista tai hiiren napin painallusta jossakin käyttöliittymän osassa, jossa hiiren painallukseen ei muulloin reagoida. Muutoin ohjelma suoritetaan pysähtelemättä alusta loppuun.

Käyttäjä voi suorittaa ohjelman keskeytyksettä painamalla työkalupalkin *Fast forward* -nappia tai valitsemalla toiminnon *Continue without stopping* valikosta *File*, jolloin ohjelma suoritetaan loppuun pysähtelemättä. Kommentit kirjoitetaan oman asetuksensa mukaan, mutta muu kuin virhetilanne tai näppäimistösyötteen pyytäminen ei keskeytä suoritusta. Virhetilanteista ilmoitetaan kommentti-ikkunassa, minkä jälkeen keskeytetään suo-

ritus. Rivi, jolla virhe sattui, jää aktiiviseksi. Käyttäjä voi näin tarkastella virheeseen johtanutta tilannetta ennen suorituksen uudelleen aloittamista. Käyttäjän annettua edellä kuvaillun jatkamaan kehoittavan syötteen palataan TTK-91-ohjelman lataamisen jälkeiseen tilaan, jolloin käyttäjän on aloitettava suoritus uudelleen alusta.

Suorituksen voi myös itse pysäyttää valitsemalla toiminnon *Stop* valikosta *File* tai painamalla työkalupalkista vastaavaa nappia. Tällöin suoritus palaa alkutilaansa ja käyttäjän on käynnistettävä ohjelman suoritus uudelleen.

5.4 Animointi

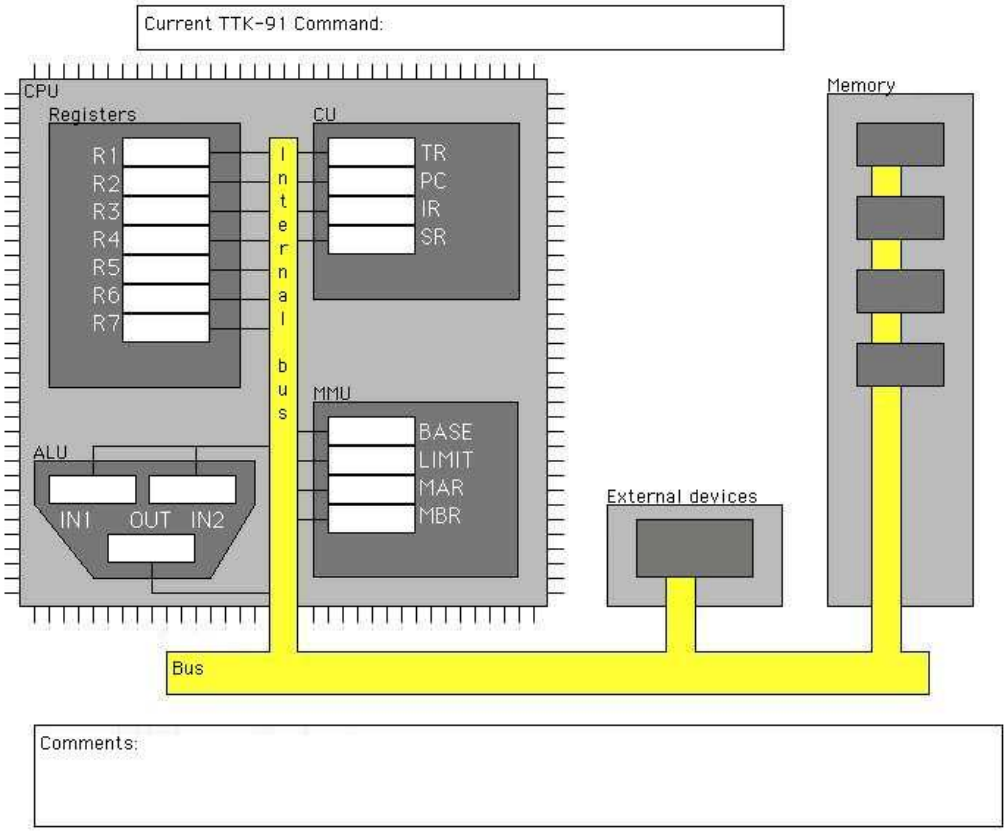
Animaattorin tarkoitus on kuvata CPU:n toimintaa rekisteritasolla sekä sisäisen väylän osalta. Tarkoitus on näyttää animoinnin avulla kuinka binäärimuotoinen konekäsky noudetaan, tulkataan ja lopulta suoritetaan. Mukana on operandien nouto ja tuloksen tallettaminen. Animointi toteutettaneen siten, että käyttäjä valitsee halutessaan suoritusmoodiksi animoidun suorituksen varsinaisen käyttöliittymän valikosta, ja ohjaa animoinnin kulkua siten, että tilasta toiseenä siirrytään käyttäjän painaessa välilyöntiä.

Animaattori kuvaa CPUn toimintaa siten, että harmaalla olevat kentät ovat muuttuvia arvoina (Rekisterit 0-7, PC TR IR ja SR Control unitissa, IN1 IN2 ja OUT ALU:ssa sekä BASE LIMIT MAR ja MBR MMU:ssa). Kuva on vain suunnitelma, mutta lopullinen toteutus sisältää samat tiedot kuin mitä tässä on esitetty. Väylän toimintaa tullaan kuvaamaan siten, että jokaisesta harmaasta kentästä on yhteys väylää pitkin jokaiseen muuhun kenttään. Aivan kuten varsinaisessa CPU:ssa väylä yhdistää jokaisen rekisterin keskenään. Memoryä ja laiteajureita ei ole kuvattu, vaan esimerkiksi muistia käytettäessä tieto ilmestyy väylältä MBR:ään kuin tyhjästä. Myöskään ALU:n sisäistä toimintaa ei ole kuvattu muuten, kuin rekisterien IN1 IN2 ja OUT osalta. Syötteet viedään IN-rekistereihin, ja tulos saadaan OUT-rekisteriin.

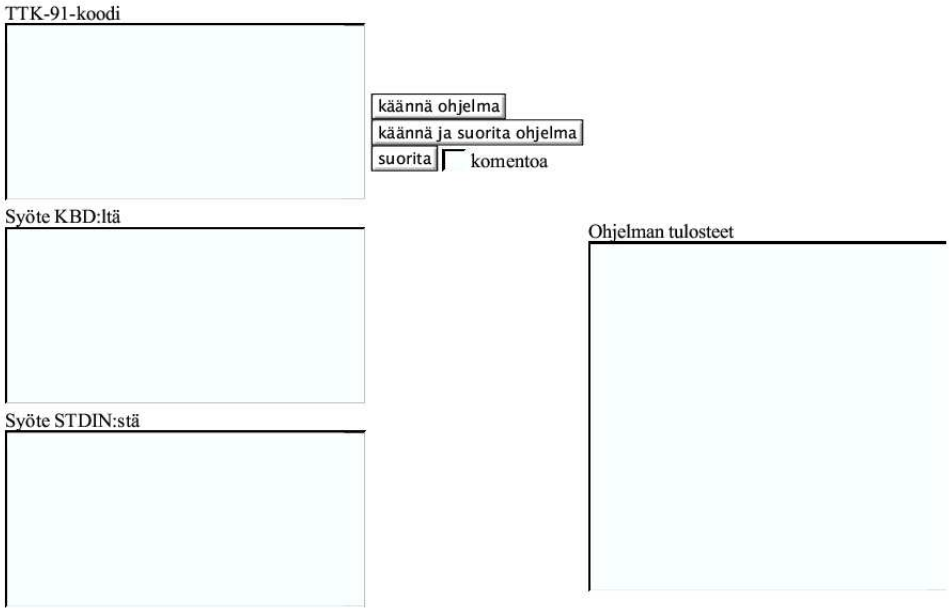
Animointi toteutetaan, jos aikaa riittää. Animointinäkymän hahmotelma näkyy kuvassa 13.

5.5 eAssari-testikäyttöliittymä

Toteutettavan eAssari-rajapinnan käytettävyyttä esitellään eAssari-demokäyttöliittymällä, joka toteutetaan HTML-kaavakkeena. Käyttöliittymästä on mahdollista kääntää ja suorittaa .k91-muodossa annettu ohjelma halutulla syötteellä joko kokonaan tai tietyn komentomäärän verran, ja tarkastella koneen tilaa toiminnon jälkeen tulostekentästä. Käyttöliittymä on hyvin pelkistetty, eikä sen yleiseen käytettävyyteen kiinnitetä enempää huomiota kuin mitä demonstrointia varten katsotaan tarpeelliseksi. Käyttöliittymä toteutetaan Java Servletinä, eikä sitä liitetä mukaan varsinaiseen opiskelijoille jaettavaan ohjelmistoon. Käyttöliittymähahmotelma näkyy kuvassa 14. Varsinainen käyttöliittymä on HTML-sivu, josta syötteet kerätään kaavakkeiden avulla.



Kuva 13: Animointinäkymän hahmotelma.



Kuva 14: Testikäyttöliittymä eAssari-rajapinnalle.

6 Keskeisten luokkien tarkat kuvaukset

Tässä luvussa esitellään eräiden luokkien tarkennettuja kuvauksia. Esittelyyn on valittu luokat, joiden toiminnallisuudet ovat sekä riittävän keskeisiä että siinä määrin monimutkaisia, että ne kaivannevat lisäselvitystä. Kuvauksissa tarkennetaan javadoc-kuvausten listausta toimintaa selittävillä vapailla tekstikuvauksilla. GUI ja GUIBrain, keskeiset käyttöliittymäluokat, on kuvattu julkisten metodien tarkkuudella. Control-luokka on valittu keskeiseksi, koska se toimii yhteistyössä, linkkinä, useiden luokkien välillä. Control-luokasta on kuvattu jokainen julkinen metodi, jotkin varsin yksityiskohtaisesti. Loader-luokka on varsin yksinkertainen, mutta se vastaa tärkeästä tehtävästä, ohjelman lataamisesta prosessorille. Processor-luokasta on kuvattu julkiset metodit. RunDebugger, Compiler ja CompileDebugger on kuvattu kontrollin etenemisen näkökulmasta, eikä niissä ole niinkään keskitytty metodittain kuvailuun.

6.1 GUI

GUI-luokka toteuttaa ohjelman graafisen käyttöliittymän. GUIBrain syöttää tälle tietoa tämän tarjoamien palvelujen kautta sekä käskyttää tätä mm. sanomalla mitkä napit tulisi olla aktiivisina ja mitkä ei.

public void actionPerformed(ActionEvent e)

Tämä metodi suoritetaan kun joku lähettää GUI:lle ActionEventin. Metodista kutsutaan sopivaa GUIBrainin palvelua.

private void initGUI()

Tämä metodi asettelee graafiset elementit paikoilleen ikkunaan.

public void setGUIView(int view)

Tämä metodi muuttaa GUI:n näkymän alku-, lähdekoodi- tai sovellusnäkymäksi aiemmin käyttöliittymäkuvauksessa mainittuun tapaan. Käytännössä ero näkyy siinä, onko vasemmassa laidassa näkyvillä kooditaulu, käsky/datataulut vaiko tyhjää. Parametrinä annetaan näkymän tunniste. Validit arvot ovat NONE, CODE_TABLE ja INSTRUCTIONS_AND_DATA_TABLE.

public void resetAll()

Tämä metodi tyhjentää näkymän syötteistä ja tulosteista.

public void unselectAll()

Tämä metodi poistaa mahdolliset valinnat kaikista ikkunan taulukoista. Valittu rivi on korostettu sinisellä. Tällä metodilla voidaan siis varmistaa ettei yksikään rivi ole korostettuna, esimerkiksi silloin, kun ohjelma siirtyy alkunäkymään (“NONE”).

public void updateStatusBar(String str)

Tämä metodi päivittää alalaidassa olevan status-palkin tekstin parametrin mukaiseksi.

public void updateReg(int reg, int newValue)

Tämä metodi päivittää rekisteritaulun halutun rekisterin arvon. Rekisterin tunnisteen tulee olla yksi määritellyistä vakioista R0:sta R6:een, SP tai FP.

public void insertToCodeTable(String[] src)

Tämä funktio korvaa kooditaulun sisällön. Kooditaulu on käytössä käännösvaiheessa, ja sisältää siis vain symbolisia komentorivejä. Parametrinä annettavan sisällön tulee olla valmiiksi muokattu siten, että jokainen rivi on omassa solussaan. Tällaisena sisältö näkyy myös ruudulla.

public boolean insertToInstructionsTable(String[] binaryCommand, String[] symbolicCommand)

Tämä funktio korvaa käskytaulun sisällön. Käskytaulua käytetään sovelluksen latauksen jälkeen, jolloin näkyvissä ovat sekä komentojen binääri- että symboliset muodot. Jokaisessa parametrissä yksi solu vastaa yhtä riviä näytöllä ja siksipä binaryCommand-parametrin tulee olla yhtä pitkä kuin symbolicCommand-parametrin, jotta joka riville saadaan molemmat vastineet. Mikäli tämä ei ole totta, funktio palauttaa arvon false eikä korvaa taulun sisältöä. Muutoin taulu korvaantuu siten, että ensimmäiseen sarakkeeseen sijoitetaan juokseva rivinumerointi, toiseen parametrin binaryCommand solut ja kolmanteen parametrin symbolicCommand solut.

public boolean insertToInstructionsTable(int[] binaryCommand, String[] symbolicCommand)

Muutoin täsmälleen samalla tavalla toimiva metodi kuin edellinen, mutta tälle annetaan parametri binaryCommand int-taulukkona, kun edellisessä sen annettiin String-taulukkona. Tämä on vain sen vuoksi, että GUIBrain tarvitsee molempia ja mikäli se joutuisi tulemaan toimeen vain toisella metodilla, niin sen täytyisi suorittaa ylimääräinen for-luuppi.

public boolean insertToInstructionsTable(String[] symbolicCommand)

Tämä metodi toimii kuten ensimmäinen, jos sille antaa ensimmäiseksi parametriksi String-taulukon, joka koostuu vain tyhjistä merkkijonoista. Muuta syytä tämän olemassaololle ei ole, kuin että se lyhentää hieman koodia.

public boolean updateInstructionsAndDataTableLine(int lineNumber, int binaryCommand, String symbolicCommand)

Päivittää käsky- tai datataulun rivin. Se kumpaa tauluista päivitetään riippuu siitä mitä riviä halutaan päivittää. Taulut mielletään tässä ikäänkuin yhdeksi tauluksi, jossa käskytaulun rivit ovat ensimmäisenä ja datataulun rivit liitettynä sen perään. Niinpä rivit 0...N päivittävät käskytaulua ja rivit N+1...N+P päivittävät datataulua, missä N on käskytaulun koko ja P datataulun. Metodi korvaa toisessa sarakkeessa olevan solun parametrin binaryCommand arvolla ja kolmannessa sarakkeessa olevan solun parametrin symbolicCommand merkkijonolla.

public boolean updateInstructionsAndDataTableLine(int lineNumber, int binaryCommand)

Muutoin toimii samoin kuten veljensä, mutta tämä ei muuta kolmannessa sarakkeessa sijaitsevaa solua.

public void insertToDataTable(String[] dataContents)

Korvaa datataulun sisällön. Ensimmäiseen sarakkeeseen tulee juokseva numerointi, joka jatkaa käskytaulun numerointia ja siksipä tätä kutsuttaesssa käskytaulun tulee olla jo luotu tai muutoin rivinumerointi ei ole yhtenäinen. Tämän ei pitäisi aiheuttaa hankalia tilanteita, sillä käsky- ja kooditaulu on ajateltu yhtenäiseksi tauluksi, jotka on vain käytännöllisistä syistä toteutettu erillisinä. Näinpä aina, jos toisen taulun sisältö voidaan muuttaa, niin myös toisen muuttamiseen tarvittava data pitäisi olla saatavilla. Käyttäjän tulee vain huolehtia, että metodit insertToInstructionsTable() ja insertToDataTable() ajetaan oikeassa järjestyksessä. Toiseen sarakkeeseen kirjataan parametrin data sisältö siten, että yksi alkio vastaa yhtä riviä. Kolmanteen sarakkeeseen tulee vain tyhjä merkkijono.

public void insertToDataTable(int[] data)

Muutoin toiminnaltaan täysin veljeänsä vastaava, mutta parametrinä annetaan String-taulukon sijaan int-taulukko, mikä aiheuttaa sen ettei tyhjää sisältöä voida tämän avulla antaa. Tämä metodi on olemassa, koska GUIBrain tarvitsee myös tätä ja vaikka se pärjäisi edelliselläkin, niin silloin se tarvitsisi ylimääräisen for-luupin muuttamaan int[] String[]:ksi.

public void insertSymbolTable(String[][] symbolsAndValues)

Tämä metodi korvaa symbolitaulun sisällön. Parametrinä annetaan kaksiulotteinen String-taulukko, jonka toinen ulottuvuus on aina kaksiulotteinen ja jonka solussa [x][0] on parametrin nimi ja solussa [x][1] parametrin arvo. Ensimmäisen ulottuvuuden koko on mielivaltainen - niin suuri kuin lisättäviä symboleja on.

public void updateRowInSymbolTable(String symbolName, Integer symbolValue)

Tämä metodi päivittää tai lisää rivin symbolitauluun. Mikäli samanniminen symboli löytyy jo valmiiksi taulusta, sen vieressä olevaa arvoa vain muutetaan eikä siis uutta riviä tarvitse lisätä. Mikäli symbolin nimen sisältävää riviä ei ole, sellainen lisätään taulukon loppuun.

public void addComment(String comment)

Tämä metodi lisää kommenttiluetteloon uuden kommentin.

public void addOutputData(int outputValue)

Tämä metodi lisää output-tekstialueeseen “näytölle kirjoitetun” uuden luvun.

public void setEnabled(short command, boolean b)

Tämä metodi aktivoi tai poistaa käytöstä halutun käskyn. Käskyjä ovat esimerkiksi "aja ohjelma", "avaa tiedosto"ym, ja niille on yleensä oma valinta valikossa tai sen lisäksi osalla oma nappi. Poikkeus on komento INPUT_FIELD, joka poistaa käytöstä tai tuo käyttöön näppäimistösyötekentän vasemmassa reunassa, sekä luvun lähettämiseen tarkoitetun napin. Komennon tunnisteen tulee olla yksi ennalta määritetyistä vakioista COMPI-

LE_COMMAND, RUN_COMMAND, STOP_COMMAND, CONTINUE_COMMAND, CONTINUE_WITHOUT_PAUSES_COMMAND tai INPUT_FIELD. Parametri b kertoo, aktivoidaanko käsky (arvona true) vai poistetaanko se käytöstä (false).

public void enable(short command) public void disable(short command)

Nämä metodit on lyhenteitä kutsulle setEnabled(command, b).

public boolean centerToLine(int line, short table)

Keskittää GUIn näkymän halutulle riville halutussa taulussa. Tämä tarkoittaa sitä, että haluttu rivi on varmasti näkyvillä tämän kutsun jälkeen. Parametrinä table voidaan antaa CODE_TABLE tai INSTRUCTIONS_AND_DATA_TABLE.

public void setSelected(short option, boolean b)

Tämä metodi muuttaa GUI:n asetuksienmuokkausnappien ja -valintojen tilaa. Sen avulla voidaan varmistaa, että toisaalta alussa GUI näyttää asetusten tilan samanlaisena kuin asetustiedostossa annettu, ja toisaalta että asetuksen muutos näkyy samanlaisena kaikkialla, vaikka sitä voidaan vaihtaa useammasta kuin yhdestä paikasta. Option-parametrin arvon tulee olla yksi ennalta määritellyistä vakioista OPTION_COMPILING_PAUSED, OPTION_COMPILING_COMMENTED, OPTION_RUNNING_PAUSED, OPTION_RUNNING_COMMENTED ja OPTION_RUNNING_ANIMATED. Parametri b kertoo, onko valinta tehty vai poistettu.

public void selectRow(short table, int row)

Tällä metodilla voidaan valita jokin rivi koodi- tai käsky/datataulusta. Käsky- ja datataulu ovat kahdessa eri ali-ikkunassa, mutta molemmat kuvaavat samaa muistia. Parametri table kuvaa taulua, josta rivi valitaan, ja sen arvon tulee olla toinen ennalta määritellyistä arvoista CODE_TABLE ja INSTRUCTIONS_AND_DATA_TABLE.

public void updateAllTexts()

Tämä metodi päivittää jokaisen ikkunassa näkyvän tekstinpätkän mahdolliselle uudelle kielelle, mikäli sellainen on valittu. Poikkeuksena vanhoja kommentteja ei käännetä, vaan ne jäävät sellaiseksi kuin olivat.

public void changeTextInEnterNumberLabel(String newText)

Tämä metodi muuttaa näppäimistösyötekentän yläpuolella olevan tekstikentän sisältämän tekstin. Jos syötteenä on tyhjä merkkijono, kenttä katoaa näytöltä.

6.2 GUIBrain

Tämä luokka on nimensä mukaisesti GUIn aivot. Se ottaa vastaan tietoa LoadInfon, CompileInfon ja RunInfon muodossa, muokkaa niistä saatavan tiedon tapahtumista GUI:lle sopivaan muotoon sekä tarjoaa sen sille näytettäväksi ruudulla. GUIBrainin tehtävä on myös kertoa GUI:lle mitä toimintoja sen tulee aktivoida ja poistaa käytöstä. Näin GUIn ei tarvitse olla tietoinen mitä GUIBrainin takana tapahtuu. GUIBrain puolestaan ei tiedä miten GUI näyttää sen tarjoamat tiedot ruudulla. Joitakin GUIBrainin metodeja kutsutaan GUIThreader-apuluokan kautta, jolloin ne pyörivät omassa säikeessään. Tästä syystä me-

todeja voidaan kutsua päällekkäin toistensa kanssa.

public GUIBrain(GUI gui)

GUIBrainin konstruktori, joka saa parametrikseen GUI-olion. GUI:n ilmentymää tarvitaan, koska GUIBrain käskyttää sitä suoraan esimerkiksi päivittäesään rekisterien arvoja, status-palkkia, ynnä muuta GUI:n sisältöä. Konstruktorissa varmistetaan, että asetukset ovat sellaiset, mitkä ne ovat asetustiedostossa tai vaihtoehtoisesti, mikäli tiedostoa ei ole tai tiettyä asetusmäärettä asetettu, ne ovat oletusarvoiset. Myös GUI päivitetään näiden asetusten mukaiseksi.

public void menuOpenFile(File openedFile)

GUI kutsuu tätä metodia, kun käyttäjä komentaa GUI:ta avaamaan tiedoston. Riippuen siitä onko valittu tiedosto b91- tai k91-päätteinen, kutsutaan Controlista joko openBinary()- tai openSource()-metodia.

public boolean enterInput(String input)

Tämä funktio syöttää Controlille näppäimistösyötteen. GUI kutsuu tätä kun käyttäjä painaa nappia, jolla annettu luku syötetään eteenpäin. Se hyväksyy vain int-muotoisen luvun, jonka pitää olla kenttien MIN_KBD_VALUE ja MAX_KBD_VALUE asettamissa rajoissa. Jos edellämainitut ehdot eivät täyty, palautetaan false eikä syötettä lähetetä eteenpäin.

public void menuRun()

GUI kutsuu tätä metodia, kun käyttäjä komentaa GUI:ta suorittamaan Titokoneen muistiin ladatun b91-ohjelman. Tällöin siirrytään suoritustilaan ja komennetaan Controlia ajamaan ohjelmaa rivi kerrallaan. Rivien välissä voidaan pitää taukoa ja odottaa käyttäjältä jatkamissyötettä, mikäli asetuksissa näin käsketään.

public void menuCompile()

GUI kutsuu tätä metodia, kun käyttäjä komentaa GUI:ta kääntämään avatun ohjelman. Tällöin siirrytään kääntämistilaan ja komennetaan Controlia kääntämään ohjelmaa rivi kerrallaan.

public void menuEraseMemory()

GUI kutsuu tätä metodia, kun käyttäjä komentaa GUI:ta tyhjentämään Titokoneen muistin. Välitetään tehtävä Control-luokalle.

public void menuExit()

GUI kutsuu tätä metodia, kun käyttäjä komentaa GUI:ta poistumaan simulaattorista. Metodi keskeyttää tämänhetkiset toiminnot, kuten käännöksen, välittömästi.

public void menuSetLanguage(String language) public void menuSetLanguage(File languageFile)

GUI kutsuu näitä metodeja, kun käyttäjä komentaa GUI:ta vaihtamaan kielen. Kielitunniste language käännetään kieliasetustiedoston tietojen avulla locale-tunnisteeksi, jolloin sen kautta komennetaan Translator-luokka päivittämään kieltään uuden localen mukaiseksi. Vaihtoehtoisesti parametrinä voidaan antaa tiedosto, jolloin itse locale jää tuntemattomaksi, ja Translatorin käsketään käyttää tiedostosta löytyvää ResourceBundlea. Tieto tal-

lennetaan asetuksiin. Mikäli tiedosto ei ole kielitiedosto, käsketään GUI:ta näyttämään siitä virheilmoitus.

public void menuSetStdin(File stdinFile)

GUI kutsuu tätä metodia, kun käyttäjä vaihtaa GUI:ssa oletusarvoisen StdIn-tiedoston. Tieto tallennetaan asetuksiin ja välitetään Control-luokalle.

public void menuSetStdout(File stdoutFile, boolean append)

GUI kutsuu tätä metodia, kun käyttäjä vaihtaa GUI:ssa oletusarvoisen StdOut-tiedoston. Jälkimmäisenä parametrina annetaan tieto siitä, ylikirjoitetaanko tiedosto joka ajon tulosteilla (append-parametrin arvo false) vai liitetäänkö uudet tulosteet vain tiedoston loppuun. Tiedosto välitetään Control-luokalle, kun taas GUIBrain pitää huolen tiedoston tyhjentämisestä tarvittaessa.

public void menuSetMemorySize(int newSize)

GUI kutsuu tätä metodia, kun käyttäjä komentaa GUI:ta muuttamaan Titokoneen muistin koon. Parametrina Controlin vastaavalle metodille välitettävä newSize on kahden eksponentti väliltä 9–16, jonka tuloksena haluttu muistin koko saadaan. Tieto talletetaan myös asetustiedostoon.

public void menuSetRunningOption(int option, boolean b)

GUI kutsuu tätä metodia, kun käyttäjä muuttaa GUI:ssa jotakin ajoaikaista asetusta. Metodi muuttaa ajoaikaista käytöstä – näytetäänkö debuggerin ajonaikaiset kommentit, suoritetaanko rivi kerrallaan ja animoidaanko suoritusta. Option-parametrin arvon tulee olla yksi ennalta määritellyistä arvoista COMMENTED, LINE_BY_LINE ja ANIMATED. Arvo b asettaa asetuksen päälle (arvo true) tai pois päältä (arvo false).

public void menuSetCompilingOption(int option,boolean b)

GUI kutsuu tätä metodia, kun käyttäjä muuttaa GUI:n käännöksen asetusta. Metodi muuttaa käännösaikaista käytöstä menuSetRunningOption-metodin tapaan. Parametrin option hyväksytyt arvot ovat COMMENTED ja PAUSED (rivittäin etenevä). Ks. menuSetRunningOption().

public void menuAbout()

GUI kutsuu tätä metodia, kun käyttäjä valitsee GUI:ssa “Help”-valikon valinnan “about”. Metodi välittää GUIlle tiedoston, joka About-ikkunassa tulisi näyttää.

public void menuManual()

GUI kutsuu tätä metodia, kun käyttäjä valitsee GUI:ssa “Help”-valikon valinnan “Manual”. Metodi välittää GUI:lle tiedoston, joka manual-ikkunassa tulisi näyttää.

public void menuInterrupt(boolean immediate)

GUI kutsuu tätä metodia, kun käyttäjä komentaa GUI:ta pysäyttämään jonkin operaation. Metodi keskeyttää tämänhetkisen toiminnon. Mikäli immediate parametrin arvoksi annetaan true, niin keskeytys on välitön eli kutsuja voi olla varma, ettei kutsun jälkeen ole muita säikeitä suorituksessa. Sen sijaan jos parametri on false, niin jokin säie saattaa jäädä vielä kertaalleen odottamaan continueTask():in suoritusta. Käytännössä GUIBrainin

itsensä ei ole koskaan järkevä käyttää non-immediate tyyppistä keskeytystä, sillä sillä kukaan ei ole aloittamassa uutta säiettä continueTask():in suorittamiseksi. Se onkin tarkoitettu GUI:n käytettäväksi esim. menuRun()-metodia suoritettaessa, kun käyttäjä on painaa stop-nappia. Tällöin kyseessä on non-immediate -tyyppinen keskeytys ja käyttäjältä odotetaan continue-napin painallusta, ennen kuin GUI:n näkymä palautetaan aloitustilaan. Uuden tiedoston avaaminen taas on esimerkki immediate- tyyppisestä keskeytyksestä - tiedoston valinta aiheuttaa aina välittömän näkymän muuttumisen, oli suorituksessa sitten mikä tahansa metodi.

public void continueTask()

GUI kutsuu tätä metodia, kun käyttäjä komentaa GUI:ta jatkamaan tauonnutta operaatiota. Se ilmoittaa odotustilaan waitForContinueTask()-metodia kutsumalla joutuneille säikeille, että operaatiota voi jatkaa.

public void continueTaskWithoutPauses()

GUI kutsuu tätä, kun käyttäjä komentaa GUI:ta jatkamaan tauonneen operaation kerralla loppuun asti. Se ilmoittaa odotustilaan waitForContinueTask()-metodia kutsumalla joutuneille säikeille, että operaatiota voi jatkaa sekä asettaa noPauses-kentän arvoksi true merkiksi siitä, ettei uusia waitForContinueTask()-kutsuja tarvitse tehdä.

public void waitForContinueTask()

Tätä metodia kutsuessaan suorituksessa oleva säie menee odotustilaan, josta vapaudutaan, kun joku toinen säie kutsuu continueTask()- tai continueTaskWithoutPauses()-metodia. Huomaa: tätä metodia ei kannata kutsua, ellei metodia ajeta GUIThreaderin kautta. Muutoin odotustilaan menevä säie on GUI:ta suorittava pääsäie ja täten GUI:n käyttö estyy.

public String[] getAvailableLanguages()

Tämä metodi palauttaa kieliasetustiedostosta löytyneet kielet. Kielelle ei tällöin välttämättä kuitenkaan ole olemassa omaa käännöstä, mutta sen Locale tunnetaan kieliasetustiedoston perusteella. Tämän perusteella voidaan päätellä, mistä tiedostosta käännöksiä tulee etsiä.

public void saveSource()

Tämä metodi tallentaa muokatun lähdekooditiedoston. Lähdekoodia voi muokata rivi kerrallaan ennen käännöstä, ja kun muokatulta riviltä poistutaan esim. rivinvaihtoa painamalal, se tallentuu sekä GUIn muistiin että levylle.

public static String getExtension(File f)

Tämä metodi palauttaa annetun tiedoston päätteen.

6.3 Control

Control on luokka, jonka kautta Titokoneen toimintaa voi tarkastella ja säädellä ulkopuolisesti, eli toisin sanoen se toimii rajapintana Titokoneen ja ulkomaailman välillä. Tässä ohjelmistossa ulkomaailmaa edustaa luokka GUIBrain, joka säätelee GUI:n toimintaa nimenomaan sen perusteella, mitä se saa Controlilta. Toisaalta GUIBrain myös käyttää Tito-

konetta Controlin kautta sen perusteella, että mitä nappeja käyttäjä on GUI:ssa painellut. Toinen ulkomaailmaa edustava luokka on yhteis-APIa testaava typistetty käyttöliittymäluokka AssariUI.

Control käyttää seuraavia luokkia:

Processor: Luokkaa käytetään koodirivien suorittamiseen.

Application: Luokka sisältää emuloitavan ohjelman, sekä ylläpitää sen crt- ja stdout-laitteille kirjoittamaa tai kbd- ja stdin-laitteilta lukemaa dataa.

Loader: Luokkaa käytetään Application-luokan sisältämän ohjelman lataamisen Titokoneen muistiin.

FileHandler: Luokkaa käytetään tiedostojen avaamiseen ja tallentamiseen. Lisäksi siltä saa .k91-tiedostosta ulos Source-olion ja .b91-tiedostosta Binary-olion.

Compiler: Luokkaa käytetään Source-olion kääntämiseen Application-olioksi.

Binary: Luokkaa käytetään Application-olioiden luomiseen Stringeistä sekä Stringien luomiseen Application-olioista.

Control tarjoaa palveluja AssariUI:lle TTK91Core-rajapintaluokan mukaisesti sekä suoraan GUIBrainille.

public TTK91Application compile(TTK91Source source) throws TTK91Exception, TTK91CompileException

Tämä metodi kääntää saamansa CompileSource-olion TTK91Application-olioksi.

- Kutsutaan Compiler-luokan metodia compile() ja annetaan sille parametrinä source String-muodossaan, jonka jälkeen kutsutaan toistuvasti compileLine()-metodia, kunnes CompileInfoja ei enää palauteta. Tämän jälkeen käännetty Application saadaan kutsumalla metodia getApplication(), jolloin se voidaan palauttaa TTK91Application-muodossa.
- Mikäli käännöksessä tapahtui poikkeus, välitetään sama poikkeus taaksepäin. Niitä ei käsitellä tässä.

public void run(TTK91Application app, int steps) throws TTK91Exception, TTK91RuntimeException

Tämä metodi suorittaa saamaansa TTK91Application-oliota steps riviä tai, mikäli steps saa arvon 0, loppuun asti. Metodi ei varaudu mitenkään mahdollisiin ikuisiin silmukoihin, vaan ylemmän luokan tulee tietää itse, millaisia rajoituksia haluaa asettaa ja antaa maksimiaskelmäärä itse.

- Pakotetaan parametrinä saatu sovellus Application-muotoon. Mikäli tämä epäonnistuu, heitetään IllegalArgumentException. Tallennetaan Application-olio myöhempää sisään- ja uloskulkevan datan tallentamista ja lukemista varten. Kutsutaan paikallista load()-metodia.

- Kutsutaan silmukassa parametrin steps määräämä määrä metodia runLine(), tai jos steps-parametrin arvo on 0, kutsutaan runLine():ä niin pitkään kunnes ohjelman suoritus päättyy.
- Mikäli tapahtuu jokin virhe, heitetään saatu poikkeus ylöspäin.

public TTK91Memory getMemory()

Tämä metodi palauttaa Titokoneen muistin TTK91Memory-oliona. Muisti saadaan TTK91-Memory-rajapintaluokan määrittävästä RandomAccessMemory-luokasta, jonka Processor palauttaa.

public TTK91Cpu getCpu()

Tämä metodi palauttaa Titokoneen prosessorin TTK91Cpu-oliona.

public String getBinary(TTK91Application app)

Tämä metodi muuttaa saamansa TTK91Application-olion binäärikoodiksi ja palauttaa sen merkkijonona.

- Luodaan luokasta Binary uusi ilmentymä ja annetaan sen konstruktorille arvoksi app pakotettuna Application-muotoon. Jos tämä ei onnistu, heitetään IllegalArgumentException. (Metodi hyväksyy TTK91Application-muotoisia olioita vain kosmeettisen API-yhtenevyyden nimissä; käytännössä toteutukset eivät osaa käyttää kuin omia sovelluksiaan.)
- Palautetaan luodun Binary-olion metodista toString() paluuarvona saatu merkkijono.

public TTK91Application loadBinary(String binary) throws ParseException

Tämä metodi yrittää muuttaa saamansa Stringin TTK91Applicationiksi.

- Yritetään luoda Binary-luokka, parametrinä konstruktorille välitetään binary-merkkijono, ja kutsutaan sen toApplication-metodia. Heitetään mahdollinen ParseException taaksepäin, muussa tapauksessa palautetaan saatu Application TTK91Application-muodossa.

public void changeMemorySize(int powerOfTwo)

Muuttaa Titokoneen muistin koon. Samalla siihen mahdollisesti ladattu ohjelma tuhoutuu, sillä muisti ja rekisterit nollautuvat.

- Tarkistetaan onko powerOfTwo sallittujen arvojen 9-16 välissä, muussa tapauksessa heitetään IllegalArgumentException. Jos arvo on sovelias, luodaan Processor-luokasta uusi ilmentymä, jolle välitetään konstruktorissa $2^{powerOfTwo}$.

public void eraseMemory()

Tämä metodi nollaa Titokoneen muistin sisällön kutsumalla Processor-olion metodia eraseMemory().

public String openSource(File openedFile) throws IOException

Tämä metodi avaa saamansa lähdekooditiedoston ja valmistautuu sen kääntämiseen.

- Tallennetaan parametrinä saatu tiedosto, jotta siitä voidaan tarvittaessa myöhemmin päätellä binäärin tallennustiedoston oletusnimi.
- Kutsutaan FileHanderin metodia loadSource() parametrillä sourceFile; mahdollinen IOException välitetään taaksepäin.
- Kutsutaan Compiler-luokan metodia compile, parametrinä FileHandlerilta saatu Source-olio String-muodossaan.

public String modifySource(String[] modifiedSource) throws IOException

Tällä metodilla voidaan muuttaa käännettävää lähdekoodia. Parametrin modifiedSource tulee sisältää lähdekoodin uusi muoto, yksi rivi per solu. Muutettu lähdekoodi tallennetaan avattuun lähdekooditiedostoon. Tiedoston tulee olla asetettu, joten tämän metodin kutsuminen ennen openSource-metodia aiheuttaa IllegalStateExceptionin. Ohjelmaa vaihdettaessa tulee kutsua openSource-metodia.

public CompileInfo compileLine() throws TTK91CompileException

Tämä metodi kääntää aiemmin alustetun tiedoston seuraavan rivin. Tiedosto alustetaan vain openSource()-metodin yhteydessä, eli useampia lähdekooditiedostoja ei voi olla samaan aikaan kääntämisvalmiina.

- Kutsutaan Compiler-olion metodia compileLine(). Välitetään heitetyt poikkeukset taaksepäin, tai onnistuneen kutsun jälkeen palautetaan saatu CompileInfo.

public void openBinary(File binaryFile) throws ParseException, IOException

Tämä metodi avaa saamansa tiedoston ja valmistautuu lataamaan sen, välittäen matkan varrella heitetyt poikkeukset taaksepäin.

- Kutsutaan FileHanderin metodia loadBinary() parametrillä binaryFile, yritetään luoda Binary-olio saadusta merkkijonosta ja kutsua saadun Binaryn metodia toApplication. Poikkeusten sattuessa ne välitetään taaksepäin.
- Tallennetaan saatu Application myöhempää sisään ja ulos kulkevan datan tallentamista ja lukemista varten.

public void saveBinary(File specialFile) throws IOException
public void saveBinary() throws IOException

Nämä metodit tallentavat avatun ja käännetyn .k91-lähdekooditiedoston binääriseksi .b91-tiedostoksi. Mikäli tiedostonimeä ei File-parametrillä ole erikseen annettu, se on sama kuin vastaavan .k91-tiedoston, mutta päätteellä “b91”.

public LoadInfo load() throws TTK91AddressOutOfBounds, TTK91NoStdInData
Tämä metodi lataa Loaderille alustetun TTK91-ohjelman Titokoneen muistiin ja siirtää joko Application-olion määrittämästä tai konstruktorissa välitetystä tiedostosta (tässä järjestyksessä) STDIN-tiedoston sisällön Application-olion lukupuskuriin. Mikäli kumpaakaan tiedostoa ei ole määritelty, tietoa ei siirretä. Application-olio tukee myös datan siirtämistä suoraan siihen, mutta edellä mainittujen tiedostojen määrittäminen yliajaa tämän datan. Mikäli Application-olion viite säilytetään ja siihen siirretään dataa tämän metodin kutsumisen jälkeen, siihen siirretyt tiedot ylikirjoittuvat.

- Asetetaan asetusten mukainen stdin/stdout-data Application-olioon lukuvalmiiksi. Tästä voi seurata alustavana varoituksena heitettävä TTK91NoStdInData-poikkeus, joka kuitenkin heitetään vasta load-metodin lopuksi, jolloin se voidaan jättää huomiotta (käyttäjälle voidaan kuitenkin näyttää ajoissa varoitus). Varsinainen kriittisempi TTK91NoStdInData-poikkeus heitetään myöhemmin sitten, jos stdin-dataa yritetään lukea, eikä lukeminen onnistu. Tällöin LoadInfon saa haettua getPendingLoadInfo()-metodilla.
- Kutsutaan Loader-olion metodia setApplicationToLoad() ja annetaan parametrinä kyseinen application.
- Kutsutaan Loader-olion metodia loadApplication() mahdollinen poikkeus välitetään taaksepäin ja saatu LoadInfo-olio palautetaan kutsujalle.

public LoadInfo getPendingLoadInfo()

Tämä metodi palauttaa LoadInfon silloin, kun load() ei “loppupoikkeuksen” heittämisen takia ole sitä ehtinyt palauttaa. Metodi palauttaa null-arvon jos lataus epäonnistui aidosti.

public RunInfo runLine() throws TTK91RuntimeException

Tämä metodi suorittaa Titokoneeseen ladattua ohjelmaa yhden rivin. TTK91RuntimeException on ajonaikaisten poikkeusten yläluokka, jonka mm. TK91NoKbdDataException (Kbd-laitteelle tarvitaan lisää syötettä) ja TTK91NoStdInDataException (StdIn-laitteelle tarvitaan lisää syötettä) perivät.

- Kutsutaan Processor-olion metodia runLine().
- Jos kutsu onnistui, niin tutkitaan RunInfoa mahdollisten Applicationin päivitysten varalta (esim. onko crt:lle kirjoitettu) ja palautetaan se lopuksi kutsujalle.
- Jos heitettiin TTK91RuntimeException, niin se vain välitetään taaksepäin, paitsi jos kyseessä oli TTK91NoKbdData tai TTK91NoFileData:

- Mikäli saadaan TTK91NoKbdData-poikkeus, kutsutaan Applicationin metodia readNextFromKbd(). Jos sieltäkin saadaan TTK91NoKbdData-poikkeus, lähetetään se taaksepäin, koska lisää kbd-dataa joudutaan kysymään käyttäjältä tai ohjelman suoritus keskeytyy. Jos readNextFromKbd() antaa jonkin luvun, annetaan se prosessorille kbd-datan syöttämiseen tarkoitetulla metodilla keyboardInput(int) ja kutsutaan runLine()-metodia uudelleen.
- Mikäli runLine() heitti TTK91NoStdInData-poikkeuksen, kutsutaan Applicationin metodia readNextFromStdin(). Jos sieltäkin saadaan TTK91NoFileData-poikkeus, lähetetään se taaksepäin. Asialle ei juuri enää voi tehdä mitään. Jos readNextFromStdIn() antaa jonkin luvun, annetaan se prosessorille stdin-datan syöttämiseen tarkoitetulla metodilla stdinInput(int) ja kutsutaan runLine()-metodia uudelleen.
- Jos suoritettavia rivejä ei enää ole, Processor palauttaa RunInfon sijaan null-arvon, joka välitetään taaksepäin. Tätä toistetaan seuraavienkin runLine()-kutsujen ajan, kunnes jokin Application ladataan seuraavan kerran.

public void keyboardInput(int inputValue)

Tämä metodi lisää arvon Processor-luokan puskuriin, josta Titokoneelle syötetään Kbd-laitteen syötteitä. Metodia kutsutaan GUIBrainista sille asti päätyneen TTK91NoKbdData-poikkeuksen seurauksena. Tätä ennen itse Application-olioon tallennettu näppäimistötieto on jo käytetty loppuun. Annettu arvo välitetään Processor-oliolle kutsumalla sen samannimistä metodia.

public void saveSettings(String currentSettings, File settingsFile) throws IOException

Tämä metodi tallentaa saamansa asetustiedot sisältävän merkkijonon tiedostoon kovalevylle. Se kutsuu FileHandlerin saveSettings()-metodia annetuilla parametreillä, ja välittää mahdollisen IOExceptionin taaksepäin.

public String loadSettingsFileContents(File settingsFile) throws IOException
public String loadSettingsStreamContents(InputStream settingsStream) throws IOException

Nämä metodit palauttavat asetustiedot kovalevyltä.

- Kutsutaan FileHandlerin metodia loadSettings() annetulla parametrillä (metodista on myös InputStream-versio).
- Palautetaan saadusta StringBufferista muodostettu String (toString()-metodilla).

public ResourceBundle loadLanguageFile(File languageFile) throws ResourceLoadFailedException

Tätä metodia tarvitaan vain kun käyttäjä valitsee oman kielitiedoston. Tavallisesti Translator-luokka löytää itse sopivaksi katsomansa kielitiedoston. Metodi palauttaa tiedoston edustaman ResourceBundle-olion, tai epäonnistuu syystä tai toisesta, heittäen kokoomapoikkeuksen ResourceLoadFailedException.

public void setDefaultStdIn(File stdinFile) throws IOException

public void setDefaultStdOut(File stdoutFile) throws IOException

Nämä metodit asettavat oletuksena käytettävät StdIn- ja StdOut-tiedostot. Niitä käytetään tiedostojärjestelmän simuloimiseen, ellei käännettävä TTK-91-sovellus itse ole määrittänyt omia tiedostojaan.

public File[] getApplicationDefinitions()

Tämä metodi palauttaa kolmipaikkaisen File-taulukon, jonka arvoina ovat käsittelyssä olevan Application-olion asetukset STDIN-, STDOUT- ja HOME-tiedostoille (viimeistä ei käytetä tässä sovelluksessa mihinkään), tai null niille arvoille joita Application-olio ei itse määrittele.

6.4 Loader

Tämä luokka lataa TTK91Application prosessorin kautta Titokoneen muistiin. Ohjelma ladataan muisiin kerralla, eli riittää että loadApplication()-metodia kutsutaan yhden kerran. Tietoa latausoperaatiosta palautetaan LoadInfo-olion muodossa, josta käyttäjä voi kysyä tarvitsemiaan tietoja.

public Loader(Processor processor)

Tämä metodi luo Loaderin parametrinä annetulle prosessorille.

public void setApplicationToLoad(Application application)

Tämä metodi asettaa ladattavan ohjelman. Vain yksi ohjelma voi olla ladattuna tai ladattavana kerrallaan.

public LoadInfo loadApplication()

Tämä metodi lataa setApplicationToLoad()-metodilla asetetun ohjelman Titokoneen muistiin. Metodi palauttaa LoadInfo-olion, joka sisältää tietoa latausoperaatiosta. Mikäli applicationia ei ole asetettu, niin metodi palauttaa arvon null. Sama ohjelma voidaan ladata useampaan kertaan tällä metodilla.

6.5 Processor

Processor-luokan tarkoituksena on simuloida TTK91-prosessorin toimintaa. Luokka pitää yllä tilatietoa suorituksen etenemisestä.

Processor käyttää seuraavia luokkia:

RandomAccessMemory: Luokkaa käytetään koneen keskusmuistin simuloimiseen.

Registers: Luokkaa käytetään prosessorin sisäisten rekisterien tallettamiseen.

RunDebugger: Luokkaa käytetään Processor-luokan viestin tulkitsemiseen käyttöliittymälle. RunDebugger generoi Processor-luokalta saatujen tilatietojen pohjalta RunInfo-olioita, jotka palautetaan GUIBrainille.

BinaryInterpreter: Luokkaa käytetään aina STORE-käskyjen yhteydessä; STORE hakee muistiriville symbolisen esitysmuodon.

Binary: Luokkaa käytetään Application-olioiden luomiseen Stringeistä sekä Stringien luomiseen Application-olioista.

TTK91RunTimeException: Luokkaa käytetään ajonaikaisen poikkeustilan ilmoittamiseen; Processor ilmoittaa Control-luokalle poikkeuksen. TTK91RunTimeExceptionin aliluokkina ovat seuraavat Exception-luokat:

- TTK91InvalidService. Tuntematon palvelukutsun (SVC) parametri.
- TTK91AddressOutOfBounds. Muistiviite muistialueen ulkopuolelle.
- TTK91IntegerOverflow. Kokonaisluvun ylivuoto.
- TTK91NoKbdData. Tällä exceptionilla Processor ilmoittaa Control-luokalle, että näppäimistödataa pitäisi saada. Control pyytää näppäimistödataa Application-luokalta.
- TTK91InvalidOpCode. Tuntematon komento.
- TTK91InvalidDevice. Tuntematon laitemäärittely. Heitetään esimerkiksi luettaessa näytöltä tai kirjoitettaessa näppäimistöön. Myös määrittelemättömän laitteen käyttöyritys aiheuttaa poikkeuksen.
- TTK91NoStdInData. Processor ilmoittaa Control-luokalle, että standardi syöttövirtadataa ei ole saatavilla.
- TTK91DivisionByZero. Nollalla jako.
- TTK91BadAccessMode. Heitetään jos käytetään määrittelemätöntä osoitusmoodia. Sallitut osoitusmoodeja vastaavat lukuarvot ovat 0 (’ ’), 1 (’=’) ja 2 (’@’).

Processor-luokan tarjoamat palvelut:

RunInfo runLine()

Tämä metodi suorittaa seuraavan komennon. Se lähettää RunDebuggerille tilatietoa. RunDebugger palauttaa rivistä RunInfo-olioita, jotka palautetaan eteenpäin.

void runInit(int initSP, int initFP)

Tätä metodia kutsutaan lataamisen lopuksi, ei siis varsinaista suoritusta aloitettaessa. Se asettaa samalla data-alueen ja koodialueen loput rekisteriarvoin.

int getValueOf(int registerID)

Tämä on TTK91Cpu-rajapinnan määrittelemä funktio; registerID viittaa määriteltyihin rekisterien arvoihin, esimerkiksi PC:n tunnus on 203 ja R5:n 406. Funktio palauttaa annetun rekisterin arvon.

int getStatus()

Tämä on TTK91Cpu-rajapinnan määrittelemä metodi; palautettava arvo määritelty samoin rajapinnassa. Metodi palauttaa ajotilan; onko suoritus kesken (STATUS_STILL_-RUNNING, arvo 901), onko ohjelma suoritettu loppuun asti (STATUS_SVC_SD, arvo 902) vaiko keskeytyikö suoritus virheen takia (STATUS_ABNORMAL_EXIT, arvo 903).

TTK91Memory getMemory()

Tämä metodi palauttaa muistin TTK91Memory-muodossa.

MemoryLine getMemoryLine(int row)

Tämä metodi palauttaa tietyn muistirivin.

void eraseMemory()

Tämä metodi tyhjentää muistin.

void memoryInput(MemoryLine inputLine, int rowNumber) throws TTK91Address-OutOfBounds

Tämä on Loader-luokkaa varten tarvittava metodi, joka kirjoittaa annetulle muistiriville annetun tiedon.

void keyboardInput(int kbdInput)

Tämä metodi täyttää Processorin näppäimistödatapuskurin (koko 1). Puskuri tyhjenee luettaessa, ja kun sitä yritetään lukea komentoa suoritettaessa tyhjänä, heitetään TTK91-NoKbdData-poikkeus.

void stdinInput(int stdinInput)

Tämä metodi täyttää Processorin tiedostolukupuskurin (koko 1). Puskuri tyhjenee luettaessa, ja kun sitä yritetään lukea komentoa suoritettaessa tyhjänä, heitetään TTK91No-StdInData-poikkeus.

6.6 RunDebugger

RunDebugger toimii prosessorin ja GUI:n välisenä tiedon viejänä. Se luo RunInfo-objekteja joihin se on kerännyt prosessorilta saamansa tiedot rivin sen hetkisestä suorituksesta sekä kommentoi mitä tapahtui. Kun prosessori luodaan, se luo samalla itselleen RunDebugger-olion. Kun prosessori saa käskyn suorittaa rivi, se ilmoittaa RunDebuggerille siitä ja suorituksen aikana ja päätteeksi se kertoo RunDebuggerille mitä on tapahtunut. Kun Run-Debugger saa tiedon siitä, että uuden rivin suoritus on alkanut, se luo uuden RunInfo-objektin johon se alkaa kerätä tietoja siitä, mitä prosessori on tehnyt ja samalla muodostaa kommentin siitä, mitä tapahtui. Kun prosessorilta tulee tieto siitä, että rivin suoritus on lopetettu, RunDebugger viimeistelee RunInfo-objektin ja palauttaa sen prosessorille ja jää odottamaan uuden rivin suoritusta. Prosessorin suorittaessa käskyä kutsutaan RunDebuggerin metodeja seuraavassa järjestyksessä. Tietojen perusteella RunDebugger päivittää tarvittavat tiedot RunInfo-ilmentymään.

1. **void cycleStart(int lineNumber, String lineContents)**

Processor ilmoittaa tällä metodilla seuraavan käskyn alkamisesta RunDebugger-luokalle. Parametreina välitetään rivinnumero ja käskyn symbolinen esitysmuoto. Metodi luo uuden RunInfo-ilmentymän ja välittää RunInfon konstruktorille parametreina saamansa arvot.

2. **void runCommand(int command)**

Processor välittää tällä metodilla konekielikäskyn int-parametrina. RunDebugger pilkkoo käskyn kaksoispistemuotoon bittisiirtoja hyväksikäyttäen. Metodi syöttää RunInfo-ilmentymälle seuraavat tiedot:

- komennon lukuesityksen
- ensimmäisen operandin, rekisterin numeron
- indeksirekisterin numeron
- osoiteosan
- muistinoutojen määrän
- kaksoispiste-esityksen sellaisenaan

RunDebuggerin generoimaa kommenttiriviä varten tallennetaan metodissa muistinoudon tyyppi String-esityksenä erilliseen muuttujaan. Myös kaksoispiste-esitys käskystä talletetaan.

3. **setValueAtADDR(int value)**

Processor kertoo tällä metodilla RunDebugger-luokalle ensimmäisestä muistinoudosta saadun arvon. Metodi asettaa saadun arvon RunInfo-ilmentymään.

4. **setSecondFetchValue(int value)**

Jos Prosessorin suorittama käsky käyttää epäsuoraa muistiviittausta, tämän metodin avulla välitetään toisen muistinoudon palauttama arvo vastaavasti kuin metodilla setValueAtADDR.

5. **setOperationType(int opcode)**

Processor asettaa operaation tyypin; luku edustaa ennalta sovittua tyyppiluokitusta.

- 0 - NOP
- 1 - Tiedonsiirtokäskyt
- 2 - ALU-operaatiot
- 3 - Vertailuoperaatiot
- 4 - Haarautumiskäskyt

- 5 - Aliohjelmakutsut
- 6 - Pino-operaatiot
- 7 - SVC-operaatiot

RunDebugger asettaa RunInfo-ilmentymälle operaation tyyppinumeron.

6. Käskyn tyypistä riippuvat metodit:

a) Tiedonsiirto: **void setIN(int deviceNumber, int value)**, **void setOUT(int deviceNumber, int value)**

Tiedonsiirtokäskyissä asetetaan setIn ja setOut -metodeilla laitteen tunnusnumero ja arvo, joka kirjoitetaan tai luetaan.

b) ALU: **void setALUResult(int result)**

setALUResult asettaa ALU-operaatiosta saadun tuloksen.

c) Vertailu: **void setCompareResult(int whichBit)**

SetCompareResult asettaa int-arvon, joka kertoo, mikä vertailubitti asetetaan: 0 - greater, 1 - equal, 2 - less.

d) Haarautuminen tai aliohjelma: **void setNewPC(int newPC)**

SetNewPC asettaa rivinumeron, jonne edetään seuraavassa käskyssä.

e) Pinoon ei liity erityisiä metodeja, vaan toiminta hoidetaan muistiasetuksilla ja rekisterien arvojen päivityksellä.

f) SVC: **void setSVCOperation(int operation)**, **void setOUT(int deviceNumber, int value)**, **void setIn(int deviceNumber, int value)**

SetSVCOperation asettaa RunInfo-ilmentymälle int-arvon mukaisen SVC-operaation tyypin. SetOUT asetetaan kun kirjoitetaan näytölle, ja setIN kun luetaan näppäimistöltä.

7. **void setRegisters(int[] registers)**

Tämän metodin avulla Processor-luokka välittää RunInfolle rekisterien arvot. Ne päivitetään automaattisesti käyttöliittymässä jokaisen käskyrivin suorituksessa.

8. **RunInfo cycleEnd()**

Processor ilmoittaa tällä metodilla komennon suorituksen päättyneen. Metodi palauttaa RunInfo-olion asetettuaan kommentit kutsumalla yksityistä metodiaan setComments. CycleEnd kutsuu myös yksityistä metodia setCompareOperation, jolla asetetaan vertailubitin arvo. Jos arvoa ei aseteta Processor-luokan käsittelemässä komennossa, asetetaan runInfo-ilmentymään RunDebugger-luokassa säilytetty aiempi vertailubitin arvo; näin vertailun tilan säilyminen välitetään RunDebugger-luokassa jokaiselle ilmentymälle.

6.7 Compiler

Kääntäjä tarkistaa annetun ohjelman kahteen kertaan. Ensimmäisellä kierroksella kääntäjä etsii koodista symbolien nimet, hoitaa muuttujien varaukset, käsittelee kääntäjän omat

valekäskyt, kuten STDIN:in määräävän käskyn, ja tarkistaa syntaksin. Ensimmäisen kierroksen jälkeen koodista poistetaan myös kommenttirivit ja kääntäjän omat ohjauskoodit, jotka eivät kuulu varsinaiseen, käännettyyn ohjelmaan. Rivillä voi olla muuttujan määrittelevä DC, DS tai EQU, rivin nimeävä tunnus tai muuttuja. Tunnus sijaitsee rivin alussa ja on muotoa ALKU LOAD R1, R2, jossa ALKU on tuo mainittu tunnus. Muuttuja voi olla varattu sana, kuten vaikkapa HALT ohjelman päättävässä käskyssä, tai käyttäjän itse määrittelemä, korkeintaan kahdeksan merkin mittainen arvo. Muuttuja pitää määritellä, mutta määrittely voi tapahtua myös käytön jälkeen. Eli ensimmäisellä kierroksella ei voida tarkistaa, onko muuttuja validi vai ei. Siksi tarvitaan kaksi käännöskierrosta.

Toisella kierroksella kääntäjä tulkkaa jokaisen käskyn symbolisesta konekieliesityksesta kokonaisluvuksi. Tämä tapahtuu seuraavasti: Rivin alusta poistetaan mahdollinen tunnus, sillä se ei käänny ohjelman binäärimuotoon. Käskystä eristetään operaatiokoodi ja muut mahdolliset osat kuten rekisterit ja osoite ja osoitusmoodi. Osat kääntyvät kokonaisluvuiksi ja niistä kasataan 32 bitin mittainen binääriesitys, joka käännetään kokonaisluvuksi konekielen määrittelyn mukaisesti. Binääritiedostot olivat alkuperäisessä Koksissa eri formaatissa kuin mitä prosessorin lukujen käsittely antaa ymmärtää (2:n komplementti), eikä lukuesitystä oltu erikseen määritelty. Tässä toteutuksessa katsottiin tärkeämmäksi täsmätä prosessorin käytöksen kanssa, jolloin myös SHRA-komennon lisäyksestä on enemmän hyötyä, joten Koksin “binääritiedostot” (.dmp) eivät tämän suhteen täsmää negatiivisilla luvuilla Titokoneen kanssa. Ohjelma käännetään muuten kuten kielen määrittely vaatii.

Kääntäjä sallii ylimääräiset välilyönnit sekä tabulaattori-merkit, mutta muu, kieleen kuulumaton roska aiheuttaa koodin hylkäämiseen johtavan virhetilanteen. Kääntäjän on myös muunnettava koodi siten, että rivien numerointi onnistuu oikein. Mikäli alussa on kaksi valekoodia vaikkapa stdin- ja stdout-tiedostojen määräämiseksi, pitää ensimmäisen oikean koodirivin olla rivillä yksi, eikä rivillä kolme.

Kääntäjän kriittisin vaihe lienee syntaksitarkistus. Kääntäjän täytyy huomata, että LOAD R1, =R2 ei ole sallittu komento, sillä rekistereiden osoitteita ei ole määritelty. Myös monissa muissa käskyissä on parametreja rajattu enemmän kuin normaalissa käskyssä. Kääntäjä palauttaa rivin käännettyään CompileInfo-olion, joka kertoo käskyn sekä käännettynä että kääntämättömänä, jotta käyttöliittymä osaa näyttää käskyn symboleineen. Lopuksi on saatavilla myös Application-olio getApplication-metodin avulla.

Kääntäjä myös huomioi varsin vaatimattomasti alkuperäisessä toteutuksessa dokumentoidut muistiosoituksen muutokset siten, että esim LOAD R1, R2:ssa ei ole yhtä muistinoutoa, vaikka käskystä puuttuu =-merkki. Myös STORE-käsky ja hyppykäskyt vähentävät yhden muistinoudon. Kääntäjä hyväksyy myös määrittelyn vastaisen, mutta edellisen version hyväksymän LOAD R1,R2-muodon, jossa välilyöntiä ei ole pilkun jälkeen. Myös muoto LOAD R1, @ R2 on sallittu.

6.8 CompileDebugger

CompileDebuuger toimii GUI:n ja kääntäjän välisenä linkkinä luoden kääntäjän käskyjen mukaan CompileInfo-objekteja, joihin se on kerännyt tiedot kääntäjän sen hetkisen tilan mukaan. Kääntäjä kertoo omista tekemisistään CompileDebuggerille kutsuen sen eri

metodeja, ja niiden mukaan CompileDebugger luo uuden CompileInfo-objektin jokaista kääntäjän kääntämää riviä kohden. Lisäksi se liittää CompileInfoon kommentin, jonka muoto noudattaa yleisesti Koksi-ohjelman kommentointia.

Ensimmäisessä vaiheessa kommentoidaan löytyneitä viitteitä, symboleita ja kääntäjän valekäskyjä. Näistä kommentoidaan oliko kyseinen viite, symboli tai valekäsky uusi vai löytyykö se jo symbolitaulusta. Vaiheen lopussa kommentoidaan symboleille varattuja muistipaikkoja. Tällä kommentti on muotoa

Varataan symbolille X muistipaikka 20.

Toisessa vaiheessa kommentoidaan käännettyjä rivejä muodossa

IN R1, =KBD -> 3:1:0:0:1.

Tässä kommentissa alkuosa on käskyn symbolinen esitys, joka löytyy riviltä. Nuolen jälkeinen osa on ositeltu käskyn binaari-esityksestä. Ensimmäinen luku on käskyn koodi, toinen on ensimmäinen operandi, kolmas on muistinosoitusmoodi, neljäs on indeksirekisteri ja viimeinen on osoiteosa. Tärkeää on huomata kääntäjän mainitsemat erikoistapaukset, joissa osoitusmuodon arvoa vähennetään yhdellä.

Kun kääntäjä luodaan, se luo itselleen CompileDebugger-olion ja kun se saa käskyn kääntää rivi, se ilmoittaa tästä CompileDebuggerille kertomalla mikä käännöksen vaihe on menossa sekä muita parametrejä sen mukaan, mitä tapahtuu.

Ensimmäisessä vaiheessa kääntäjä kertoo CompileDebuggerille, mikä oli rivin numeron ja oliko rivi tyhjä, tai rivin symbolisen sisällön. Tämän jälkeen kääntäjä tutkii rivin sisällön ja ilmoittaa CompileDebuggerille, löytyikö riviltä symboleita tai viitteitä ja määriteltiinkö ne viitteiden tapauksessa rivillä vai lisättiinkö ne symbolitauluun. Lisäksi symbolit ja viitteet kommentoidaan joko tuntemattomiksi tai tunnetuiksi. Ensimmäisen vaiheen lopussa kääntäjä määrittelee löytyneiden symbolien arvot ja välittää ne CompileDebuggerille.

Kun kääntäjä kutsuu finalFirstPhase-metodia, se välittää tiedon siitä, minkälainen muistin sisältö on kun siitä on poistettu tyhjät rivit. CompileInfossa välitetään myös symbolitaulu sekä data-alue, jonka lopussa on mahdolliset DEF-käskyt.

Toisessa vaiheessa kääntäjä välittää CompileDebuggerille tiedon siitä, mitä käännettävä rivi sisältää. Se ilmoittaa CompileDebuggerille, mikä oli rivin symbolinen sisältö sekä käännetty binääri. Mukaan liitetään merkkijonona binääriesitys, johon on paloiteltu binaariesitys omiksi osikseen. Osat esitetään kokonaislukuina ja ne on eroteltu toisistaan kaksoispistein.

CompileDebuggerille välittää nämä tiedot sitä mukaa CompileInfo-olioon kun se saa niitä. Käännösoperaation lopuksi se lisää infoon commentin sekä tilanneviestin jossa kerrotaan mikä vaihe oli menossa, esimerkiksi “Syntaksitarkistus.” Tämän jälkeen se palauttaa muodostetun CompileInfo-olion kääntäjän kautta GUIBrainille. GUIBrainilla käyttää heuristiikkaa, jonka avulla se osaa hakea oikeat tiedot CompileInfosta eri muuttujien mukaan.

6.9 BinaryInterpreter

BinaryInterpreter hoitaa käännöksen binääriformaattia kuvaavasta kokonaisluvusta symboliseen käskymuotoon. Tämä käännös ei ole täysin triviaali sillä eri käskyt sisältävät eri määrän parametrejä ja osassa käskyistä suoriteteen eri määrä muistinoutoja kuin binääriformaatti antaisi ymmärtää.

Yleisesti käskyt tulkitaan muotoon: OpCode Ri, M ADDR (Rj) missä OpCode on käskyn operaatiokoodi, Ri on ensimmäinen operandi, M on muistinoutoa vastaava symboli, ADDR on osoiteosa ja Rj suluissa on toinen rekisteri. Esimerkiksi LOAD R1, =1 kääntyy muotoon LOAD R1, =1(R0). Tämä on suora käännös binääriformaatista symboliseen. Tästä on kuitenkin joitakin poikkeuksia.

Jokainen muistin rivi tarkistetaan josko se sisältäisi jonkin käskyn. Tällöin oikeiden komentojen muistiriveille kirjoitetaan kyseisen komennon symbolinen vastine josta on jätetty pois turhat parametrit säilyttäen kuitenkin indeksirekisterin.

Tässä tarkistuksessa jokainen muistin rivi jonka operaatiokoodi on nolla eli luku on pienempi kuin kaksi potenssiin 24, tulkitaan NOP-käskyksi. Kuitenkin tulkittaessa käskyistä tulostetaan vain olennaiset parametrit eli NOP:n perään ei kirjoiteta mahdollisia parametrejä koska näillä ei ole vaikutusta NOP:n suoritukseen. Kuitekin poikkeuksena on virheellinen muistinoutojen määrä kolme, jolloin käsky hylätään ja riville kirjoitetaan luvun perään tyhjä jono.

Käskyssä STORE sekä kaikissa hyppykäskyissä binääriformaatissa on näiden käskyjen muistinoutojen määrä oikea mutta se tulkitaan yhtä suuremmaksi. Tällöin jos STORE käskyn binääriformaatissa lukee nolla muistinoutoa, se tulkitaan symboliksi “”. Vastaavasti yhtä noutoa vastaa 1 eli “” joka tulkitaan näiden käskyjen tapauksessa @-merkiksi.

Käskyn 16-bittiä sisältävä osoiteosa tulkitaan etumerkilliseksi yhden komplementissa olevaksi kokonaisluvuksi. Tällöin luvut 0-32767 tulkitaan suoraan positiivisiksi luvuiksi. Luku 32768 on ns. miinus nolla. Luvut 32769-65534 ovat negatiivisia lukuja aina lukuun -32767. Tähän konversioon on käytetty kaavaa: if(luku>=32768) luku=luku-32768; luku=luku*(-1); Tästä seuraa kaksi nollaa. Tämä käännös suoritetaan näin siksi, että koodit olisivat yhteensopivia vanhalle KOKSI-ohjelmalle kirjoitettujen ohjelmien kanssa.

7 Ohjelman ulkoiset määritykset

Tässä luvussa sidotaan toteutuksen yksityiskohtia, kuten hakemistorakenne ja eri tiedostojen sijainnit sekä asetustietojen ja käännöstiedostojen muoto.

7.1 Tiedostot ja hakemistorakenne

Projektin luokat kuuluvat paketteihin `fi.hu.cs.titokone` ja `fi.hu.cs.ttk91`. Jälkimmäinen paketti sisältää yhteis-API:n luokat, jotka luetellaan liitteenä 3 olevassa dokumentissa. Ohjelmisto toimitetaan levitykseen jar-pakettina, jota voi ajaa sellaisenaan

tai avattuna. Ohjelmiston tiedostot on sijoitettu tämän paketin sisällä seuraavasti:

README.txt kuvaa lyhyesti paketin sisällön ja viittaa asennusohjeisiin.

INSTALL.txt kertoo, kuinka sovellus asennetaan ja kuinka se käynnistetään.

Makefile helpottaa luokkatiedostojen kääntämistä 'make'-komennon avulla.

javadoc/ sisältää `javadoc`-työkalulla automaattisesti generoidut luokkakuvaukset; javadoc-hakemistossa on index.html, josta pääsee käsiksi muihin html-tiedostoihin, jotka on sijoiteltu tämän hakemiston lisäksi alihakemistoihin fi/hu/cs/titokone/, fi/hu/cs/titokone/resources/ ja fi/hu/cs/ttk91/.

fi/hu/cs/titokone/ sisältää projektin java- ja class-tiedostot muutamia poikkeuksia lukuunottamatta.

fi/hu/cs/titokone/resources/ sisältää java- ja class-tiedostot käännösluokille.

fi/hu/cs/titokone/doc/ sisältää käyttöohjeen ja TTK91-koneen spesifikaation.

fi/hu/cs/titokone/etc/ sisältää oletusasetukset tiedostossa titokone.cfg ja kieliasetukset tiedostossa languages.cfg.

fi/hu/cs/titokone/img/ sisältää käyttöliittymän tarvitsemat kuvatiedostot.

fi/hu/cs/ttk91/ sisältää java- ja class-tiedostot yhteis-API:n luokille.

Kun käyttäjä muuttaa asetuksia, hänen kotihakemistoonsa (java.lang.System.getProperty ("user.home")) tallennetaan tiedosto titokone.cfg, joka vastaa rakenteeltaan oletusasetustiedostoa.

7.2 Asetukset ja kielitiedostot

Eri asetustiedostojen rakenne on keskenään hyvin samankaltainen. Simulaattorin asetukset sisältävä titokone.cfg sisältää riveittäin avain- ja arvopareja, jotka on erotettu "="-merkillä. Avain ei saa sisältää tätä merkkiä, eikä arvo-osa saa sisältää rivinvaihtoa (\n, \r tai \n\r). Muuten näiden avain-arvo-parien muotoa ei ole rajoitettu. Oletuksena asetustiedosto kuitenkin sisältää tiedot ajo- ja käännösasetuksista (riveittäin vai ei, kommentoiden vai ei, ajon tapauksessa animoiden vai ei), muistin koosta (mikä kahden potenssi on kyseessä), valitusta kielestä (tiedostonimi URI-muodossa, joka viittaa käyttäjän itse osoittamaan käännöstiedostoon, tai merkkijono, joka viittaa kieliasetustiedostoon) sekä StdIn- ja StdOut-tiedostojen nimet. "#"-merkillä alkavat rivit ja tyhjät, vain välilyöntejä tms. sisältävät "whitespace"-rivit jätetään huomiotta. Ajo- ja käännösasetuksissa kullakin asetuksella on tietty arvo, joka lisätään tulokseen, mikäli asetus on päällä. Kommentoinnin arvo on 1, riveittäin etenemisen 2 ja animoinnin 4. Kommentoiden ja riveittäin etenevä ajo tai käännös olisi siis asetusarvoltaan $1 + 2 = 3$. Esimerkkiasetustiedosto voisi näyttää tältä:

```
# Default settings. Running mode: commented, one row at a
# time, compilation mode: commented. Memory size 2^9 = 512
# words. Stdin and stdout file paths are relative to the
# current working directory. Stdout use describes that the
# file should be used in append mode instead of overwriting
# it for every new run.
Language = English
Running mode = 3
Compilation mode = 1
Stdin file = stdin
Stdin path = relative
Stdout file = stdout
Stdout path = relative
Stdout use = append
Memory size = 9
```

Kieliasetustiedosto sisältää samanlaisia pareja, joissa avaimena on kielen paikallinen nimi ja arvona merkkijono, josta voidaan lukea kieltä vastaava koodi (“locale”): pisteellä erotettuna ISO-639-standardin mukainen kielikoodi, ISO-3166-standardin mukainen valtiokoodi ja käyttöympäristön spesifioiva muunnelmakoodi. Vain ensimmäinen osa on pakollinen, ja joko muunnelma- tai valtio- ja muunnelmakoodit voidaan jättää pois. Tällainen merkkijono voisi siis olla esimerkiksi “fi”, “en.GB” tai “es.ES.MAC”, “es.ES.WIN” tai “es.ES.POSIX”. Näiden mukaisesti käännösluokkaa etsitään edellä kuvatusta resources-hakemistosta tiedostosta Translations_fi.class, Translations_en_GB.class tai tarvittaessa Translations_en.class, Translations_es_ES_MAC.class tai Translations_es_ES.class tai Translations_es.class. Oletuksena on aina Translations.class, jos muita ei löydy. Tämä luokka sisältää sovelluksen oletuskielisen käännöksen, englanniksi. Kieliasetustiedosto voisi näyttää esimerkiksi tältä:

```
English = en.GB
Suomi = fi.FI
```

Itse kielitiedostot ovat java.util.ListResourceBundle-luokan laajennoksia, joita haetaan resources-hakemistosta niiden tiedostonimen perusteella, kuten edellä kuvailtiin.

7.3 Käyttöjärjestelmän vaikutus simulaattorin toimintaan

Vaatimuksissa kerrotaan, että simulaattorin tulee toimia sekä Windowsissa että Linuxissa. Käytännössä perussimulaattorin toiminnalle ei ole kuin yksi rajoitus, kunhan kelvollinen Javan virtuaalikoneen implementaatio on saatavilla. Kelvollinen virtuaalikoneen implementaatio tarkoittaa tässä yhteydessä Java2-versioita 1.4.2:sta eteenpäin. Koski-ryhmä ei takaa simulaattorin yhteensopivuutta aiempien Javan virtuaalikoneiden kanssa.

Rajoituksen aiheuttavat hakemistopolut: koska joihinkin asetustiedoston ja TTK-91-koodin (valekäsky DEF) kohtiin voidaan tallentaa myös absoluuttisia tiedostopolkuja (lähinnä

STDIN- ja STDOUT-tiedostojen sijaintiin), todettakoon että emme voi taata esimerkiksi Windowsin levyasemasymboleiden (“C:\”) tulkintaa järjestelmissä, joissa hakemistojärjestelmä eroaa (kuten Linux). Jos absoluuttisen polun muunto käyttökelpoiseksi toisessa järjestelmässä ei onnistu Javan palveluiden avulla, oletuksena käytetään työhakemiston stdin- ja stdout-tiedostoja. Asetustiedostossa ja binäärissä hyväksytään myös polut suhteessa työhakemistoon, ja näitä varten asetustiedostossa on selvennykseksi oma muuttuja, jonka arvo on joko “relative” tai “absolute”. Mikäli polku on “relative”, kaikki hakemistoerottimiksi tulkittavat merkit korvataan järjestelmän omilla hakemistoerottimilla ja tiedosto etsitään suhteessa työhakemistoon. Järjestelmä pyrkii itse tallentamaan hakemistoerottimensa sopivan yleismaailmallisessa muodossa asetustiedostoihin, kuten esimerkiksi Javan URL-luokan toString-tulosteena.

8 Vaatimusten huomiointi suunnittelussa

Vaatimusmäärittelydokumentissa lueteltiin ja analysoitiin asiakkaan vaatimukset. Samalla kullekin vaatimukselle annettiin siihen viittaava numero, joita tässä käytetään vaatimuksiin viittaamiseen.

8.1 Toiminnallisten tavoitteiden saavuttaminen

[V1] Suunnittelun tavoitteena on ollut selkeä ja helppokäyttöinen ohjelmisto, joka soveltuu hyvin tito-kurssin tarpeisiin. Tämä tavoite on pyritty saavuttamaan käyttöliittymän suunnittelulla, joka on esitelty luvussa 5.

[V2] Vaatimus ohjelman taaksepäin yhteensopivuudesta: Vanha Koksi suoritti myös spekseistä poikkeavia ohjelmia, mistä seuraa ristiriitainen tilanne, missä pitää valita kunnioitetaanko TTK-91-koneen määrittelyä vai vanhan Koksin toimintaa. Päätettiin, että ensisijaisesti toteutuksemme seuraa TTK-91-koneen määrittelyä. Tämä tarkoittaa siis sitä, että määritelmien kanssa ristiriitaiset vanhassa Koksissa toimineet ohjelmat eivät tule toimimaan tässä toteutuksessa. Yksi esimerkki tällaisesta käskystä on “LOAD R1,=R2”.

Lisäksi TTK-91-koneen määrittelystä puuttuvat, mutta vanhassa Koksissa osittain toteutetut käskyt NOT, ROL, ROR, ASL ja IRET jätetään toteuttamatta. Kyseiset käskyt menivät vanhassa Koksissa kääntäjästä läpi, mutta suorituksessa ne olivat tuntemattomia käskyjä.

[V3] Vaatimus vanhan semantiikan noudattamisesta: Operaatiokoodien ja operaatioiden toiminnan kuvauksen pohjana on käytetty vanhan Koksin speksejä, joten ne ovat yhtenevät.

[V4] Tuki Windows- ja Linux-käyttöjärjestelmille: Javan Swingillä toteutettu käyttöliittymäprototyyppi toimii sekä Windows- että Linux-ympäristössä.

[V5] Vaatimus erillisistä palasista, vaatimus kääntäjän toiminnasta sekä käyttöliittymän kielestä: Valitsin, kääntäjä, lataaja ja suoritin ovat suunnittelussa jaettu omiin luokkiinsa. Tosin kääntäjä ei ota syötteenä merkkijonoa vaan yhteisen API:n mukaisen TTK91Source-olion, josta saadaan merkkijonoesitys. Samoin kääntäjä palauttaa TTK91Application-

olion, ei suoranaisesti .b91-olioita. Kyseiset olion saadaan palautetulta TTK91Application-oliolta.

Simulaattorin käyttöliittymän kieltä voidaan vaihtaa ja käyttäjä voi generoida uusia kieliä. Oletuskielenä on englanti.

[V6] Vaatimus simulaattorin suorituksen graafisesta toiminnasta: Käyttöliittymä näyttää päivittyneet rekisterien ja muistin arvot, jokaisen käskyn suorittamisen jälkeen.

[V7] Vaatimus koodi- ja datasegmentin erottelusta: Käyttöliittymäprototyypissä muisti on automaattisesti erotettu koodi- ja datasegmenttiin. Käyttäjä voi vierittää molempia erikseen mielensä mukaan.

[V8] Vaatimus koodieditorin poisjättämisestä: Koska vaatimus oli lähinnä muotoa “tätä ei tarvita”, totesimme testausvaiheessa, että koodin muokkaaminen käännösyrityksen jälkeen käyttöliittymässä olisi joskus hyödyllistä. Lisäsimme riveittäin muokkausmahdollisuuden ennen kääntämistä mm. pienten syntaksivirheiden korjaamiseksi. Varsinainen lähdekoodin muokkaus oletetaan silti tehtäväksi muualla.

[V9] Vaatimus dokumentoinnista englanniksi: Javadoc-dokumentointi on toteutettu englanniksi. TTK-91-koneen määrittely on käännetty englanniksi. Käyttöohjetta ei ole vielä kirjoitettu. Puuttuva dokumentaatio toteutetaan testaus- ja loppusiivousvaiheen aikana.

[V10] Vaatimus koodin lukemisen helppoudesta: Koodista on pyritty tekemään helppolukuista. GUI-, GUIBrain- ja Processor-luokat ovat kuitenkin toiminnallisuuden hajautusyrityksistä huolimatta varsin laajoja, ja etenkin GUI-luokan koodia tuskin kannattaa lukea luennolla – graafiset käyttöliittymät vaativat melkoista määrää pitkiä ja monimutkaisia kutsuja eri käyttöliittymäolioihin, eikä koodista ole siksi helppo tehdä salonkikelpoista. Metodien ja kenttien nimet on valittu kuvaaviksi mahdollisuuksien mukaan.

[V11] Tuki eAssari-rajapinnalle: Kääntäjän sekä simulaattorin syötteet ja tulosteet eivät ole suoraan merkkijonomuotoisia, kuten vaatimuksissa kuvailtiin. Sen sijaan eAssari-tukea varten on suunnitelu molemmille ryhmille asiakkaan hyväksymä yhteinen rajapinta. Tämän API on suunniteltu enemmän tai vähemmän ryhmien yhteistyönä ja se on jäädytetty. Työmme toteuttaa API:n kuvaaman rajapinnan, ja tätä kautta tuleva eAssari-ohjelmisto hyödyntää palvelujamme. “Binääriformaatin” toteutuksesta ei ollut yksiselitteistä spesifikaatiota (ks. Keskeisten luokkien tarkat kuvaukset, Compiler) negatiivisten lukujen suhteen, joten ryhmien toteutus voi vaatia yhdenmukaistamista.

[V12] Tuki itseään muuttavalle ohjelmalle: Simulaattori tukee itseään muuttavan koodin näyttämistä koodialueella. Jos koodialueelle kirjoitetaan jokin arvo, simulaattori tarkastaa vastaako arvo jotakin symbolista konekielikäskyä ja jos vastaa, niin tämä symbolinen käsky näytetään. Tämä pätee myös data-alueelle kirjoitetuille arvoille.

[V13] Muistirivien esitystapa: Käyttöliittymässä konekäskyn kymmenkantainen numeerinen arvo on näkyvissä symbolisen käskyn rinnalla koodi- ja data-alueella. Heksa- ja binäärimuotoja ei näytetä.

[V14] Tuki hakemistopuulle tiedostoa avatessa: Ohjelman valinta toteutetaan Javan valmiilla tiedostodialogilla, joka osaa hyödyntää koko hakemistorakenteen.

[V15] Vaatimukset asetuksista: Muistin kokoa ja käyttöliittymän kieltä voidaan vaihtaa

simulaattorin käyttöliittymällä. Muistin koko on aina jokin kahden potenssi, välillä 512-65536. Muistin kokoa muuttavaa valekäskyä ei toteuteta, mistä sovittiin asiakkaan kanssa. Levytiedostoja voi vaihtaa sekä käyttöliittymästä että valekäskyllä DEF.

[V16] SHRA-käskyn lisääminen käskykantaan: SHRA-käskyn oikeanlainen toiminta vaatii, että kokonaisluvut esitetään 2:n komplementtimuodossa. Tätä vaatimusta ei ole esitetty vanhan TTK-91-koneen spesifikaatiossa, joten tämä vaatimus on lisätty uuteen versioon TTK-91-koneen spesifikaatiosta.

[V17] Vaatimus levyoperaatioiden simuloimisesta: Levyoperaatioita simuloidaan IN- ja OUT-käskyillä. IN-käskyllä luetaan STDIN-tiedostosta ja OUT-käskyllä voidaan kirjoittaa STDOUT-tiedostoon.

[V18] Vaatimus suoritussyklin animoinnista: Animoinnin vaatima informaatio käskyn suorituksen vaiheista tallennetaan suorituksesta kertovaan objektiin (RunInfo). Itse animaatioikkunan pohjana käytetään tito-luentomonisteen mukaista kuvaa TTK-91-koneen keskeisestä sisällöstä. Kuva sisältää rekisterit, ALU:n, MMU:n, kontrolliyksikön, muistin ja väylät. Animointi havainnoillistaa näiden yksiköiden välistä toimintaa.

[V19] Vaatimus laiteajurin animoinnista: Laiteajurien toiminnan animointia ei toteuteta.

8.2 Laadullisten tavoitteiden saavuttaminen

[V20] Siirrettävyys käyttöjärjestelmästä toiseen: Simulaattori toteutetaan Java- kielellä, jonka pitäisi toimia useissa eri ympäristöissä. Käyttöliittymää testataan Linuxilla sekä Windowsilla ja ainakin prototyyppi toimii näissä käyttöjärjestelmissä. Ohjelman jakelu toteutetaan JAR-pakettina.

[V21] Tuki kahdelle käyttöliittymärajapinnalle: Molempien Koksi-ryhmien yhteinen rajapinta mahdollistaa TTK-91-ohjelmien kääntämis- ja suorittamispalvelujen hyödyntämisen selkeästi dokumentoidulla tavalla.

[V22] Kattava dokumentointi: Suunnitteluvaiheessa dokumentointityö on aloitettu sisällyttämällä toteutettaviin luokkiin Javadoc-dokumentointi. Lisäksi käyttöohje ja TTK-91-koneen määrittely dokumentoidaan testausvaiheen aikana.

[V23] Joustavuus: Muistin koko on vaihdettavissa ohjelman käyttöliittymästä miksi tahansa kahden potenssiksi viidestä kuuteentoista, jolloin muistin koko sijoittuu välille 512-65536. Kieli voidaan vaihtaa käyttöliittymässä mihin tahansa määriteltyyn kieleen. Oletuskieli on englanti. Lisäksi käyttöliittymä mahdollistaa ohjelman suorituksen joko käsky kerrallaan tai kerralla loppuun asti, kommenteilla tai ilman.

[V24] Testattavuus: Testaukselta on vähennetty viikko projektisuunnitelman alkuperäisestä arviosta aikataulun venymisen takia. Silti testaamiseen jää aikaa kolme viikkoa, mikä riittänee ohjelman toiminnan riittävään varmistamiseen.

[V25] Oikeellisuus: Toiminnalliset tavoitteet 1-17 toteutetaan. Lisäksi animointia koskeva tavoite 18 toteutetaan, jos aika riittää, mutta laiteajurin toimintaa koskevaa tavoitetta 19 ei toteuteta. Kaikki laadulliset tavoitteet, siis tavoitteet 20-29, toteutetaan.

[V26] Luotettavuus: Tämä vaatimus täytetään muiden vaatimusten kautta.

[V27] Tehokkuus: Suunnittelussa tehdyissä ratkaisussa ei ole pyritty huomioimaan suorituksen tehokkuutta vaan toiminnan oikeellisuutta.

[V28] Itsesuojelukyky: Kaikki mahdolliset TTK-91-ohjelman suorituksessa kysyttäessä palautettavat tiedot ovat mahdollisuuksien mukaan kopioita varsinaisen koneen tilasta, eivätkä viitteitä, jotta näiden kyselytietojen avulla ei voida muuttaa varsinaisen koneen tilaa. Koska kopiointi ei kuitenkaan aina ole käytännöllistä, joskus joudutaan yhteis-APIn toteuttamiseksi palauttamaan olioita, jotka voidaan pakottaa takaisin varsinaiseen muotoonsa. Esimerkiksi TTK91Core-luokka palauttaa TTK91Memory-olion, jota voi vain lukea, mutta jos sen pakottaa RandomAccessMemory-muotoon, muokkaus onnistuu.

[V29] Käytettävyys: Käyttöliittymäsuunnittelulla on pyritty saavuttamaan helppokäyttöinen ohjelma. Käyttöliittymän prototyyppiä on käytettävyystestattu muutamalla koehenkilöllä.

Liite 1. Luokkien Javadoc-kuvaukset

Liite 2. TTK-91-spesifikaatio

Liite 3. Yhteinen API-spesifikaatio

Liite 4. Esimerkkiohjelma B91-muodossa

```
___b91___           (varattu sana, 3 alaviivaa molemmin puolin)
___code___          (varattu sana)
0 9                 (koodialueen alku ja loppu eli init FP)
52428801
18874378
572522503
36175883
287834122
18874379
538968064
36175883
69206016
1891631115
___data___          (varattu sana)
10 11               (data-alueen alku ja loppu eli init SP)
0                   (data-alue)
0
___symboltable___   (varattu sana)
luku 10             (symbolitaulu, vain paikalliset symbolit)
summa 11            (kääntäjän symb. mukana oletusarvoisesti)
___end___           (varattu sana)
```