Osaamista
ja oivallusta
tulevaisuuden
tekemiseen

Matti Kauppila

Helppo koodieditori

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Tieto- ja viestintätekniikka

Insinöörityö

14.4.2020

Tekijä Otsikko Sivumäärä Aika	Matti Kauppila Helppo koodieditori 30 sivua 14.4.2020
Tutkinto	Insinööri (AMK)
Tutkinto-ohjelma	Tieto- ja viestintätekniikka
Ammatillinen pääaine	Pelisovellukset
Ohjaaja	Lehtori Miikka Mäki-Uuro

Insinöörityössä oli tarkoituksena tuottaa yksinkertainen koodinkäsittelyohjelma. Koodinkäsittelyohjelman tarkoitus oli olla muistinkäytöltään kevyt ja käyttökokemukseltaan mahdollisimman helppo. Projektin vaatimusmäärittely tehtiin omatoimisesti, koska työllä ei ollut erillistä tilaajaa. Vaatimusmäärittelyyn vaikuttivat suuresti jo olemassa olevat koodieditorit ja niiden ominaisuudet. Tarkoitus oli poimia kaikista eri editoreista parhaat ominaisuudet ja tuoda ne kaikki samaan ohjelmistoon.

Työ koostui pääasiassa ohjelmakoodista ja koodiin liittyvistä kaavioista. Ennen ohjelmakoodin toteuttamista piirrettiin luokkakaavio havainnollistamaan tarvittavia luokkia. Ajankäytöllisesti suurin osa ajasta meni ongelmatilanteiden selvittämiseen, uusien ominaisuuksien luomisen sijaan.

Työ toteutettiin avoimen lähdekoodin Debian Linux -alustalla. Se kirjoitettiin C++-ohjelmointikielellä, ja ongelmatilanteita etsittiin Valgrind-apuohjelmalla. Ohjelmakoodin kääntämiseen käytettiin Makefile-skriptejä. Versiohallinta tehtiin manuaalisesti ilman versionhallintaympäristöä.

Tuloksena syntyi lähes käyttövalmis tekstieditori, joka voi toimia alustavana pohjana mahdolliselle jatkokehitykselle. Ohjelmalla voi ladata, muokata ja tallentaa ohjelmakooditiedostoja. Täyden ohjelmointiympäristön luonti on yhdelle ihmiselle liian suuri tehtävä muutamassa kuukaudessa. SDL2-kirjaston avulla on mahdollista luoda Windows- ja Linux-versiot ohjelmasta automaattisesti.

Avainsanat	editori, C, C++, IDE, sulautettu kehitysympäristö

Author Title Number of Pages Date	Matti Kauppila An easy code editor 30 pages 14 April 2020
Degree	Bachelor of Engineering
Degree Programme	Information and Communications Technology
Professional Major	Game Applications
Instructor	Miikka Mäki-Uuro, Senior Lecturer

The purpose of this study was to produce a simple code editing software. The goal was to create a light-weight and cross-platform editor without unnecessary plugins. The aim for this final year project originated from personal experiences across different editors. The goal was to gather all good features from the field and compose a simplistic code editor.

Only open source software were used when writing the thesis. Most of the code was written with Vim text editor and the code was compiled with GNU compiling tools, namely G++ and Make. Memory leaks were debugged with the Valgrind memory leak detecting software. SDL2 makes it possible to compile K-DI across various platforms and computer architectures.

As a result of this study, a simple text editor was created. Due to time constraints, it was not possible to create all the planned functionalities. The editor is now able to append characters to text, save and load files and respond to specific operating system events.

Keywords	Editor, C, C++, IDE, Integrated Development Environment

Sisällys

1. Johdanto

Insinöörityön tarkoitus on luoda uusi ohjelmakoodieditori. Editorilla ei ole ulkopuolista tilaajaa. Tavoitteena on saada ohjelman toiminnan kannalta välttämättömimmät ominaisuudet toimimaan. Insinöörityössä tuotetaan ainoastaan prototyyppi. Lopullisena tavoitteena on editorin julkaisu GitHub-palvelussa,

K-DI (*koodi)* on ohjelmoijan kirjoitusväline ja työkalu. Markkinoilla on kymmenittäin vaihtoehtoisia koodieditoreja, mutta ne kaikki vaihtelevat paljon ominaisuuksiltaan. Nykytrendeinä ohjelmistokehityksessä ovat alati kasvavat tiedostokoot, käyttötietojen raportointi ja kerääminen sekä automaattiset ohjelmistopäivitykset. Tämä näkyy myös nykyisten ohjelmointiympäristöjen valtavina tilavaatimuksina ja jatkuvana yhteytenä Internetiin. K-DI on diskreetti työkalu, joka välttää laitteiston ja verkon ylimääräistä kuormitusta,

K-DI-editoriin on tarkoitus ottaa mukaan toiminnallisia ominaisuuksia, jotka ovat tuttuja monista IDE-työkaluista (*Integrated Development Environment)*, kuten pikanäppäimet ohjelmakoodin kääntämiselle, hierarkianäkymä, projektitiedostot ja suora pääsy käyttöjärjestelmän konsoliin. Suomeksi puhutaan sulautetuista kehitysympäristöistä. Laajentamisen varaa on, mutta projekti ei saa juuttua liikaan monimutkaisuuteen. Kohderyhmänä ovat pääasiassa kokeneet ohjelmoijat, vaikka ohjelmaa suunnata vain yhdelle ryhmälle. Myös ihmiset, jotka eivät halua lähettää telemetriatietojaan kolmansille osapuolille, otetaan huomioon suunnittelussa. Väri- ja ulkoasuvalinnat valitaan herättämään nostalgisia tunteita käyttäjässä.

Tuettavia ohjelmointikieliä ovat C ja C++. Tulevaisuudessa K-DI-editoriin tulee tuki myös muille yleisimmille ohjelmointi- ja skriptikielille. Luvussa 5.2 hahmotellaan myös JavaScript - ja Java -analysaattorit. Projektin kantavat periaatteet ovat läpinäkyvyys, helppokäyttöisyys ja luotettavuus. Toisin kuin valtaosa olemassa olevista tietokoneohjelmista, K-DI:n ei ole tarkoitus päivittää itseään käyttäjän tahdon vastaisesti eikä lähettää käyttäjän telemetriatietoja eteenpäin. Automaattinen tekstinsyöttö on oletuksena pois päältä. Tavoitteena on luoda selkeä tekstieditori, joka toimii Mac OS X:ssä, Linuxissa ja Windowsissa, tuottamatta ylimääräistä vaivaa käyttäjälle. Tuettavat prosessoriarkkitehtuurit ovat x86, x64 ja ARM. Työn tulosta on

tarkoitus hyödyntää ohjelmistokehityskäytössä. K-DI ja sen lähdekoodi tulevat ladattaviksi työn julkaisun jälkeen Internetiin sen omilla kotisivuille.

Luvussa 2 puhutaan suosituimmista kehitysympäristöistä yleisesti. Toteutettavat ominaisuudet käydään läpi luvussa 4. Suunnittelu ja toteutus selvitetään luvussa 5 ja sen alaluvuissa. Luvuissa 6–7 käydään läpi tuloksia ja pohditaan tulevia mahdollisuuksia.

2. Yleisimmät kehitysympäristöt

Tekstieditoreita on olemassa todella paljon, ja eri ohjelmistojen suosiosta on tehty pienimuotoinen tutkimus [1], jonka tuloksista saadaan kuva siitä, kuinka kova kilpailu markkinoilla on. Ohjelmoijien keskuudessa esiintyy vankkaa uskollisuutta omaa suosikkiympäristöä kohtaan, joten näkyvyyden saaminen on haastavaa. Ilmiötä kutsutaan nimellä ”Editor War” [2]. Nykyisin Emacs ja Vim eivät ole enää yleisön ainoita suosikkieditoreja, vaan rinnalle ovat nousseet kaupallinen Sublime Text ja Microsoftin Visual Studio Code. Microsoftin Codea pidetään usein nykymaailman de-facto -editorina. Gvim on maininnan arvoinen graafinen versio Vim-editorista. Tuki hiirelle, minkä Gvim tarjoaa, helpottaa todella paljon Vimin käyttöä.

Muistinkulutus on Microsoftin Visual Studiossa todella suurta. Pahimmassa tapauksessa se varaa keskusmuistia useita gigatavuja, ja käynnistymisaika ilman SSD-kiintolevyä lasketaan jopa minuuteissa. Bloatwareksi kutsutaan ohjelmistoa, joka on tarpeettoman suuri suhteessa sille määrättyyn tehtävään. Ohjelmiston kokoa mitataan sen käyttämän massa- ja keskusmuistin määrästä. Käyttäjän tietoturvaa ei voida aina taata useissa nykyaikaisissa editoreissa, koska koko tekstisyöte saatetaan välittää Internetin läpi palvelimelle. Visual Studion markkinaosuus käyttäjistä oli vuoden 2018 mittauksessa (taulukko 1) yli 28 prosenttia, eli se on tällä hetkellä maailman suosituin ohjelmistokehitysympäristö [1].

1. Käyttäjäosuudet vuoden 2018 suosituimmista editoreista [1].

Nimi	**Käyttäjäosuus**
Microsoft Visual Studio	28,43 %
Vim	16,54 %
Qt Creator	11,64 %
Visual Studio Code	10,31 %
Clion	8,91 %
Emacs	7,4 %
Xcode	4,1 %
Eclipse	4,0 %
Kdevelop	1,2 %
Sublime Text	1,4 %

Borlandin Turbo C++ [3] kilpailee samassa sarjassa K-DI:n kanssa. Sen graafinen ulkoasu on suunniteltu samalla ajatuksella, mutta Borlandin Turbo-sarjan kääntäjät ovat jo jääneet historiaan, eikä Borlandin editori QuickBasicin kaltaisesti tue näytönohjaimen grafiikkatiloja. GNU-työkalut ovat todella hyvin dokumentoituja. Niiden toiminnallisuutta pidetään aktiivisesti ajan tasalla. Esimerkkikoodia ja selvitettyjä ongelmatilanteita löytyy Internetistä todella paljon. Kaikki GNU-työkalut ovat avointa lähdekoodia.

3. Käytetyt työkalut ja kirjastot

K-DI-editoria työstäessä käytettiin avoimen lähdekoodin työkaluja. Kirjastoja tarvittiin, koska graafisen käyttöjärjestelmäikkunan luomisesta ja käyttöjärjestelmän pyynnöistä huolehtiva koodi olisi täytynyt kirjoittaa muuten jokaiselle alustalle erikseen. Ulkopuolisia kirjastoja käyttämällä pystytään luomaan graafinen ikkuna kuvan piirtämistä varten vain yhdellä esitystavalla. Alla ovat esitettynä eri työkalut lyhyen kuvauksen kera.

K-DI:n käyttämät kirjastot:

- SDL2

SDL2image
SDL2ttf.

SDL2 eli Simple DirectMedia Layer on ohjelmointikirjasto, jonka C-rajapinnalla voidaan piirtää näyttölaitteeseen grafiikkaa sekä vastaanottaa käyttöjärjestelmältä ja laitteilta erilaisia käskyjä. SDL2image lisää tuen pakatuille .png -kuville. SDL2ttf sisältää metodit ja tietorakenteet TrueType-kirjasimien lataamista ja esittämistä varten [4].

Kehityksessä käytetyt työkalut:

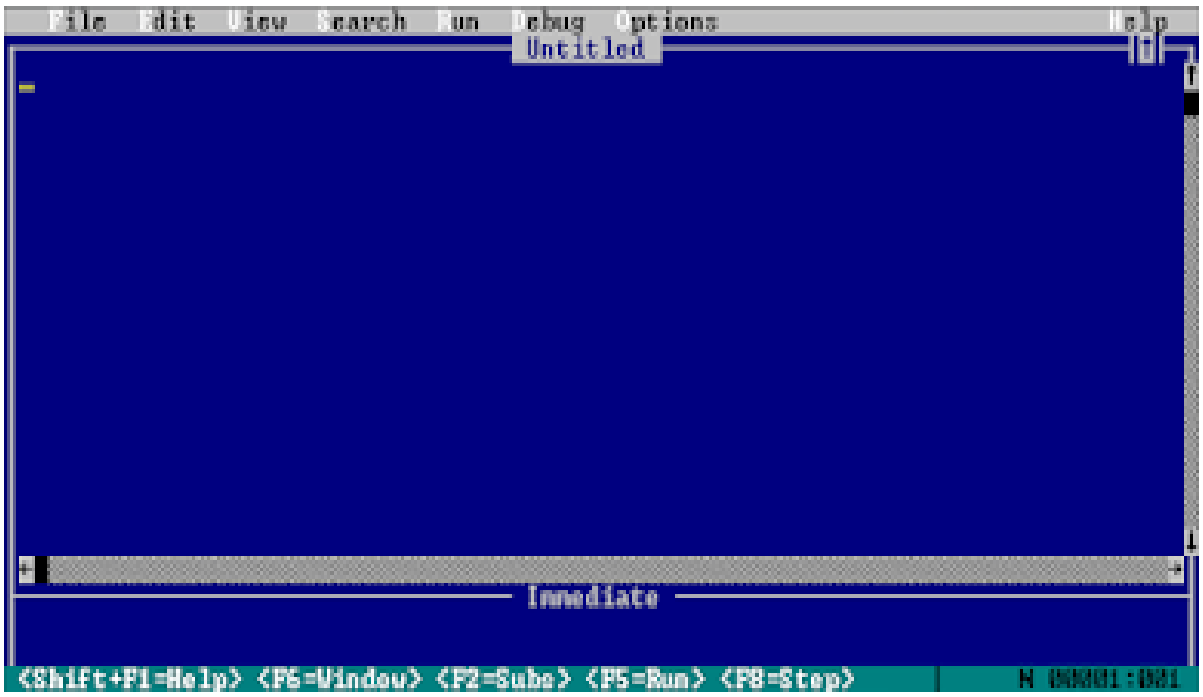
Vim
GNU Make
GNU C Compiler
Valgrind.

Ohjelmistokehityksessä käytettiin Vim-koodieditoria, joka on hyvin yksinkertainen ja suosittu editori. Osa koodista on kirjoitettu Sublime Text -editorilla. K-DI on käännetty ja suunniteltu kääntymään GNU C -ympäristössä. Käännösskripti on Makefile-formaatissa.

Valgrind [5] on tehokas työkalu muistivuotojen tutkimista varten. C++ on hyvin vaativa kieli, jossa varattu ohjelmamuisti on palautettava takaisin käyttöjärjestelmälle manuaalisesti, joten virheiden etsiminen on hyödyllistä, jottei ohjelma suorituksen aikana joutuisi ongelmatilanteeseen. Valgrindista ei ole Windows-versiota.

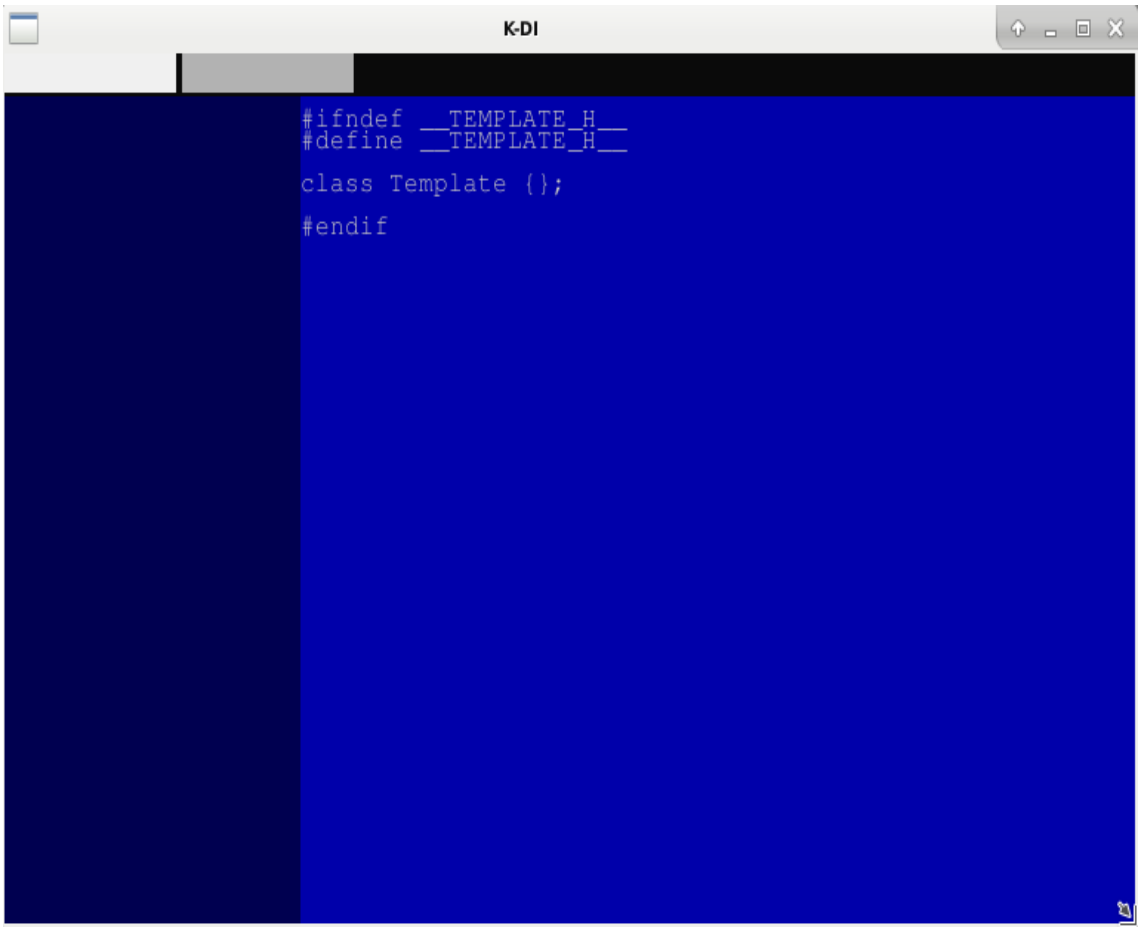
4. Ominaisuusmäärittely

Opinnäytetyönä toteutettua ohjelmistoa, K-DI:a, suunniteltiin jo vuosia sitten. Ohjelmiston valikoituminen opinnäytetyöksi antoi syytä luoda koherentin ja realistisen toteutussuunnitelman K-DI:n kehitystä varten. Ominaisuuksien rajaaminen ja lopullisten teknisten ratkaisujen valitseminen aloitettiin joulukuussa 2019. Pääasiallinen toimintatarkoitus K-DI:lla on ohjelmakoodin muokkaaminen ja ohjelmakoodikokonaisuuksien toteuttaminen. Työn esikuvana vaikutti suuresti Microsoftin vanha ohjelmointiympäristö QuickBasic (kuva 1). Nimenomaan QuickBasic on johdatellut todella monen suomalaisen ohjelmakoodin äärelle.



Kuva 1. Microsoftin QuickBasic-ohjelmointiympäristö [6].

Plagiointiin ei haluttu syyllistyä, joten oli aiheellista ottaa huomioon, ettei ihan täyttä kopiota toisen tahon tuotteesta oltu valmistamassa. Jotta ulkoasuja on helppo vertailla, kuvassa 2 on kuvankaappaus myös K-DI:sta. Väriteema on valittu tarkoituksella mahdollisimman samankaltaiseksi kuin Microsoftin tuotteessa. QuickBasic toimii täysin tekstitilassa. K-DI tukee korkeita näyttölaitteen erottelutarkkuuksia, erilaisia kirjasimia ja tuhansia värejä, joten elementtien piirtämisessä on paljon enemmän mahdollisuuksia.



Kuva 2. K-DI-editorin väriteema.

K-DI on natiivi Windows- ja Linux-ohjelma ja se tukee TrueType-kirjasimia sekä tiedostojen pudotusta suoraan sovelluksen ikkunaan. Kuvassa 2 ylhäällä näkyvät harmaat laatikot ovat välilehtiä. Välilehtien toiminnallisuus tässä prototyypissä on jätettiin toissijaiseksi, vaikkakin ne piirretään ruudulle. Jokainen välilehti toteuttaa TextField-luokan ja voi toimia puskurina tekstille. Välilehti pystyy lataamaan sisältönsä tiedostosta, joten tiedostojen lataaminen ja tallentaminen on tämän elementin vastuulla.

Ikkunan väriteemaa ja asetuksia pystyy muuttamaan joko K-DI:n konfiguraatiotiedostosta tai projektikohtaisesti projektitiedostosta. Mikäli väriteemoja ei ole määritelty ohjelman ulkopuolelle, väriteemaksi valitaan sinivalkoinen teema, joka on kovakoodattuna ohjelman sisällä. Etsimisjärjestys konfiguraatioille on moniportainen. Mikäli mitään asetustiedostoja ei löydetä etsinnän jälkeen, värit ja muut asetukset valitaan kovakoodatuista vaihtoehdoista, ja mikäli käyttäjällä on kotihakemistossaan K-DI:n konfiguraatiotiedosto, väripaletti luetaan sieltä.

K-DI pitää kirjaa projektista ja siihen kuuluvista osista projektitiedostoilla. Projektitiedoston olemassaolo on keskeisimpiä eroavaisuuksia tavalliseen tekstieditoriin. Projektitiedostot ja konfiguraatiotiedostot noudattavat samaa notaatiota.

Ominaisuusmäärittelystä laadittiin myös taulukko 2 havainnollistamaan kokonaisuutta.

2. K-DI-editorin ominaisuudet.

Ominaisuus	**Tärkeys**
Elementtien piirto ruutuun	Välttämätön
Näppäimistöohjaus	Välttämätön
Dynaaminen käyttöjärjestelmäikkuna	Välttämätön
Lähdekoodin luku ja kirjoitus	Välttämätön
Kohdistin	Välttämätön
Tunnistegeneraattori	Välttämätön
Projektitiedoston luku ja kirjoitus	Valinnainen
Ohjelmakoodin analyysi ja värjäys	Valinnainen
Tiedostojen pudotus ikkunaan	Valinnainen
Konsolinäkymä	Valinnainen
Luokkanäkymä	Valinnainen
Hierarkianäkymä	Valinnainen
Vapaa näppäimistökartta	Valinnainen

Luvussa 7 on esitelty taulukko toteutuneista ominaisuuksista.

5. Suunnittelu ja toteutus

1. Työn kulku

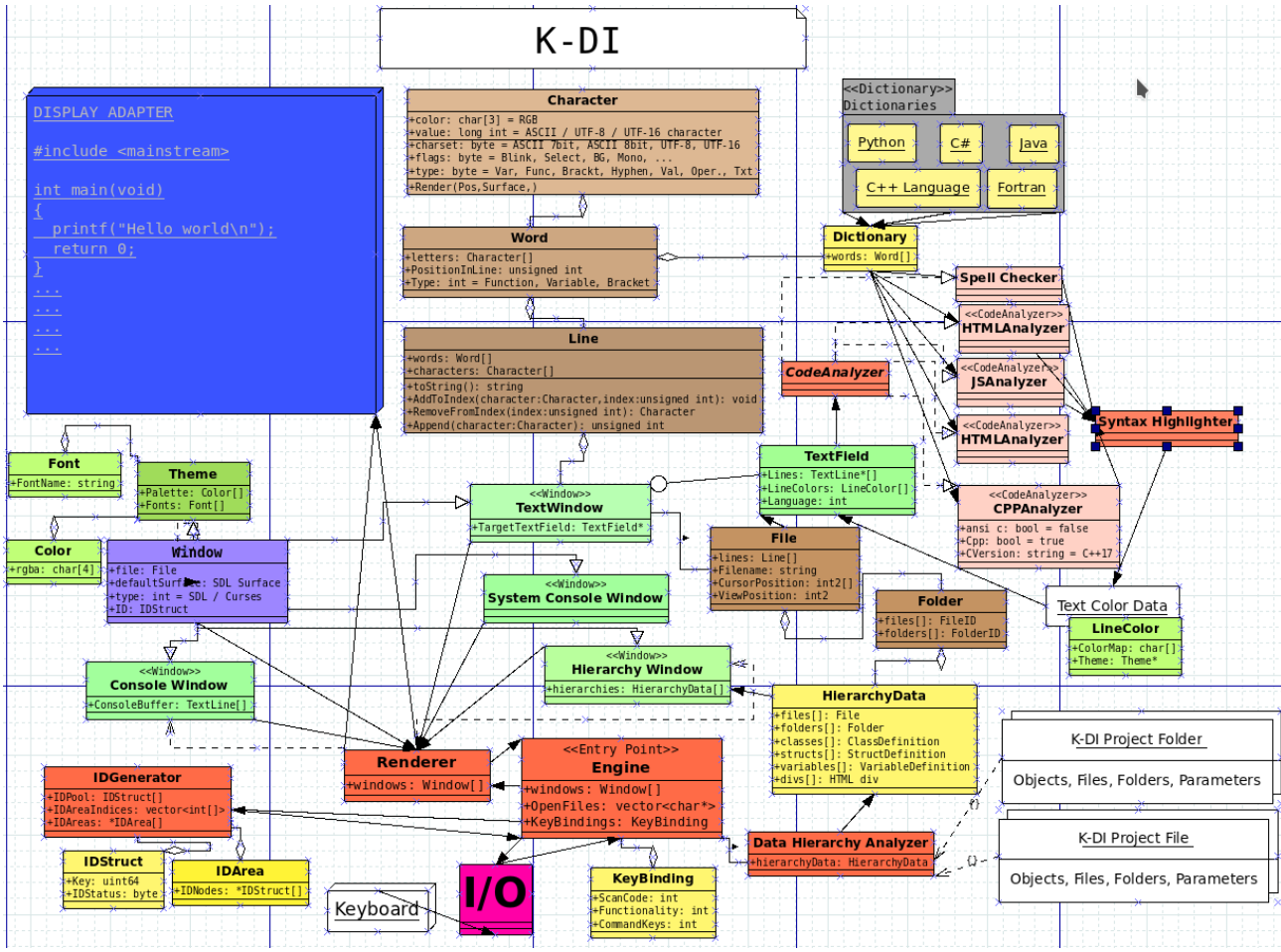
K-DI-editoria alettiin suunnitella tositarkoituksella joulukuussa 2019, koska henkilökohtainen tarve hyvälle tekstieditorille oli todella suuri. Ohjelmakoodista osa on vanhoista projekteista, joita ei ole koskaan julkaistu tai viety valmiiksi asti. Työstä pidettiin työpäiväkirjaa. Työpäiväkirjaan merkittiin tuottoisimpien päivien tulokset sekä pahimmat virhetilanteet. Virhetilanteita oli paljon, ja niistä riittää kerrottavaa.

Työn tekemisen aikana opittiin paljon uutta SDL-kirjastosta ja eri käyttöjärjestelmien yhteensopivuuksista. Makefilestä tehtiin ominaisuuksiltaan sellainen, että se kykenee itsenäisesti tulkitsemaan käytössä olevan suoritinarkkitehtuurin ja käyttöjärjestelmän. Tieto käännösympäristöstä välitetään ohjelmalle esikääntäjän parametreina. Tämä ratkaisu mahdollistaa automaattiset käännökset myös ARM-suoritinarkkitehtuurille.

Yksikkötestejä ei toteutettu lainkaan, koska työssä keskityttiin enemmänkin pelkkään välttävään toiminallisuuteen. Yksikkötestit ovat todella hyvä tapa varmistaa ja estää virhetilanteiden syntyminen ennakkoon. Projektin aikataulu ei antanut myöten suunnitella tällaista järjestelmää. C++ ei tue mitään standardia testausalustaa, vaan testit pitää rakentaa itse. Google Test [7] on Googlen tarjoama C++ -testausympäristö, joka on mahdollinen valinta K-DI:n jatkokehitykseen.

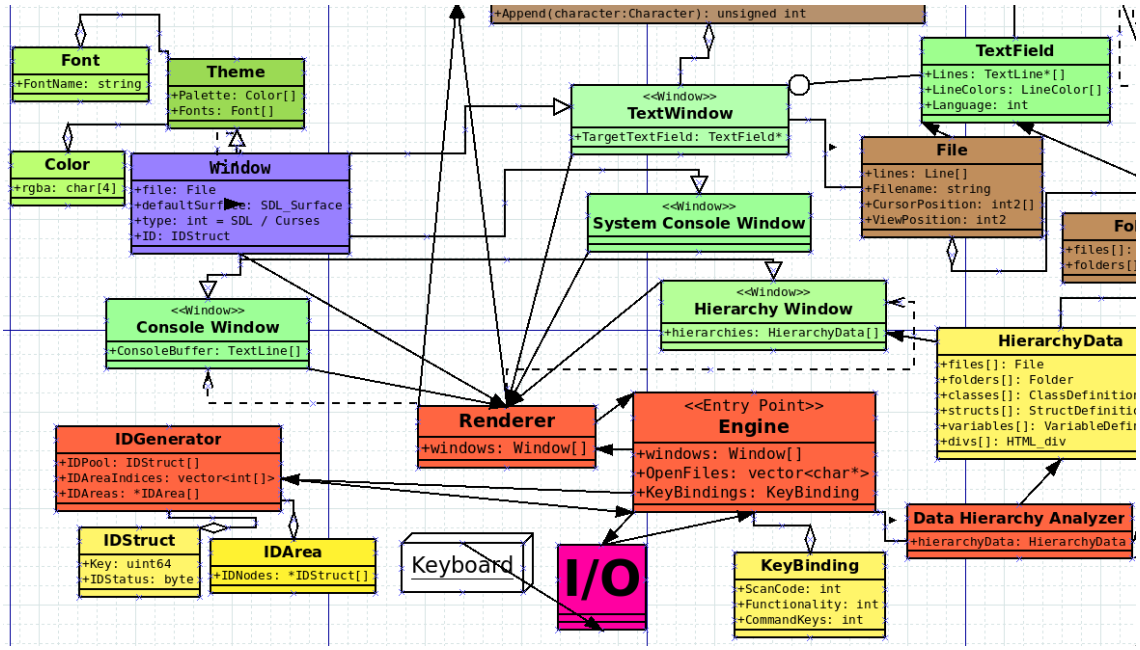
2. Luokat

K-DI:n suunnittelussa seurattiin olio-ohjelmoinnin periaatteita, ja jokainen ohjelmiston komponentti kapseloitiin omaksi luokakseen Jesse Libertyn teoksen [8] määrittämien ohjeiden mukaan. Työstä valmistettiin ensin luokkakaavio (kuva 3) havainnollistamaan eri komponentteja ja niiden välisiä suhteita. Kaaviossa mustaksi värjätyt nuolet havainnollistavat ohjelmaprosessin kulkua komponenttien välillä, katkoviivat informaatiovirtoja ja läpinäkyvät nuolet luokkien omistussuhteita.



Kuva 3. Luokkakaavio K-DI:n keskeisistä toiminnallisuuksista.

Punaisella merkityt komponentit ovat toiminnallisia osia, vihreällä merkityt visuaalisia, keltaisella merkityt ovat tietovarastoja ja ruskeat abstrakteja tietomalleja. Näyttölaite on kaaviossa merkitty sinisellä värillä ja näppäimistösyöte vaaleanpunaisella. Purppuralla värillä merkitty Window-luokka on abstrakti luokka ohjelman ikkunasta, josta periytetään kaikki muut erilaiset ikkunat, kuten TextWindow ja HierarchyWindow. Toiminnalliset elementit on esitelty tarkemmin luokkakaavion suurennuksessa. (kuva 4).



Kuva 4. Toiminnalliset komponentit K-DI-editorin luokkakaaviossa.

Punaisista komponenteista ainoastaan ”Engine” on merkitty leimalla Entry Point (kuva 4*),* eli ohjelman suoritus alkaa siitä kohdasta. Se vastaa ohjelman muistinhallinnasta ja suoritusjärjestyksestä, ja sen vastuulla on myös ohjelman pääsilmukka, joka toteutetaan metodissa MainLoop(). Pääsilmukka odottaa SDL-kirjastolta SDL_Event -tyyppisiä tapahtumailmoituksia, jotka käsitellään saapumisjärjestyksessä. Jokainen iteraatio yrittää tyhjentää tapahtumapuskurin kokonaan, joten jos tapahtumia on liikaa, ohjelma voi kokea hidastumista. Lukkiutuminen on estetty ajastimella, eli jos tapahtumapuskuria ei saada tyhjennettyä tarpeeksi nopeasti, siirrytään väkisin seuraavaan iteraatioon. Jos ohjelman suorituksessa tapahtumapuskuriin jää jatkuvasti ”velkaa”, se voi lopulta puskurin täytyttyä käyttäytyä määrittämättömästi.

Näppäimistösyötettä varten on olemassa ”KeyBinding”-luokka, jossa eri napeille on määritetty eri toiminallisuuksia. Näppäinkartta on täysin käyttäjän asetettavissa, ja mahdollisesti *SVORAK* - ja *DVORAK* -näppäimistöasettelut tulevat mukaan K-DI:iin jossain kehitysvaiheessa. Kohdistimen liikuttamiseen voi käyttää nuolinäppäimiä ja yhdistelmiä ”ghjk” ja ”wasd”, riippuen käytettävästä asetuksesta. Vim-editorin tyyliin editori voidaan asettaa useaan eri tilaan (mm. *VISUAL*, *INSERT*, *REPLACE*), joka

antaa eri näppäimille eri toiminnot eri kontekstissa. Useiden eri näppäimistöasettelujen tukemista ei prototyyppiin saatu mahtumaan mukaan.

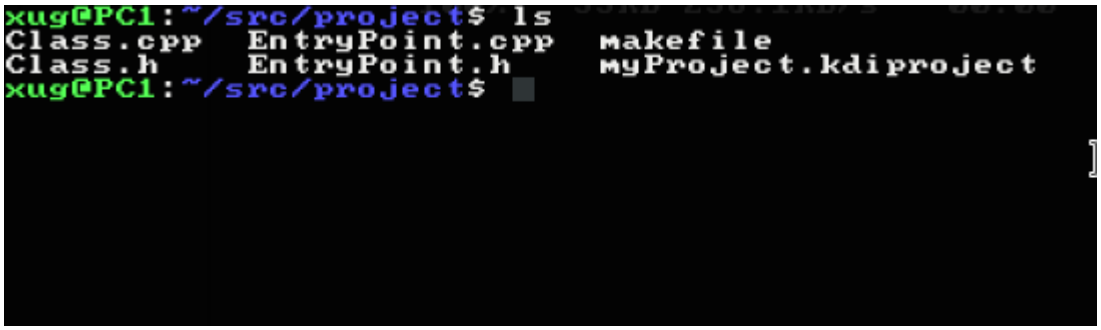
IDGenerator ratkaisee tunnisteen omalaatuisella tavallaan, koska siihen ei projektin aikana löydetty mitään varmaa tapaa uniikin ID-tunnisteen luomiseen. Yksi tapa on generoida kymmenentuhannen alkion joukko täyteen numeroita, sekoittaa sitä ja valita sieltä aina seuraava ID halutulle ohjelman elementille. Tällä hetkellä käytetään ohjelman 64-bittisen UNIX-ajan sekuntilukeman viimeistä 16:ta bittiä, joita siirretään bittioperaatiolla ”<<” 48 bittiä taaksepäin. Tämän jälkeen ID:hen lisätään C-kielen clock()-funktion palauttama arvo, eli lopullinen ID on Unix-ajan loppuosan ja prosessin syklien määrän ohjelman käynnistyksestä summa. FindDuplicateIDs()-metodi suoritetaan, ettei duplikaatteja ID-referenssejä esiinny. Mikäli duplikaatti ID löytyy, instanssille määritetään uusi ID. Elementtien numerointi tunnistepohjaisesti on välttämätöntä, koska ohjelman uudelleenkäynnistyksen yhteydessä kaikki osoittimet eri objekteihin katoavat ja ne on luotava uudestaan ID-numeroiden perusteella. ID-numeroita säilytetään projektitiedostossa.

Theme-luokka sisältää listan väreistä ja kirjasintyypeistä, joita käytetään eri tilanteissa ohjelmakoodin, valintalaatikoiden ja muun käyttöliittymän ilmaisuun. Myös osoittimen tyyppi ilmaistaan Themen sisällä. Teemoja voi muokata tekstitiedostoina ja ladata ohjelman suorituksen aikana muistiin.

SDL_Event on SDL-kirjaston tietorakenne käyttöjärjestelmän ja ikkunointijärjestelmän tapahtumien, kuten näppäimistösyötteen, hiiren liikkeen, ohjelmaikkunoiden toiminnan sekä käyttöjärjestelmän lähettämien signaalien tarkasteluun. Näppäimistösyöte otetaan vastaan ja suoritetaan ExecuteKeybinding()-metodissa, joka huomioi olemassa olevat KeyBinding-luokan ilmentymät ja suorittaa näppäimistöpainallusta vastaavan toiminnon ohjelman sisäisenä funktiokutsuna. SDL_Event mahdollistaa myös Drag&Drop -tapahtuman eli sen, että käyttäjä pudottaa tiedoston ikkunaan käyttöjärjestelmän pudotustoimintoa käyttäen.

Ennen lopullista toteutusta suunnittelussa isona apuna olivat ulkopuoliset lähteet, joissa ohjeistetaan tekstieditorin tekemiseen. Lähteenä käytetyn opetussarjan tekijä keskittyy ohjelmassaan hieman eri asioihin, kuin mihin K-DI:ssa keskitytään. Hän ei luo lainkaan konsolinäkymää eikä makroja editoriinsa [9]. K-DI-editorista pyrittiin toteuttamaan käyttäjäystävällinen ohjelmisto, jossa itse käyttökokemus on hiottu

ajatuksella ja pitkällä harkinnalla. Lisäksi IDE-maailmasta vaikutteina haettiin käteviä lisäominaisuuksia, kuten tiedostojen pudotus tekstinkäsittelyikkunaan, projektihierarkia ja integroitu konsolinäkymä (kuva 5).



Kuva 5. Konsolinäkymä Linux-järjestelmässä.

Konsolinäkymä jää vain Linux-puolen toiminnallisuudeksi (kuva 4, s. 10). Kumpikin käyttöjärjestelmä kykenee muodostamaan automaattiset käännösskriptit, mutta vain POSIX-tyypin järjestelmät tarjoavat helpon pääsyn käyttöjärjestelmän konsoliin.

Tuettuja käyttöjärjestelmiä ovat Windows, Linux ja MacOS X. Ainoa tuettu kääntäjä on tällä hetkellä GNU C -kääntäjä, josta on olemassa jokaiselle tuetulle järjestelmälle oma versionsa. Windows-järjestelmissä suositellaan käytettäväksi MinGW32/64-ympäristöä. BSD-pohjaisilla järjestelmillä ohjelma saattaa toimia, mutta takeita tästä ei ole.

Prototyyppi tukee näppäimistön lisäksi myös hiirisyötettä. Tuki hiirelle kuitenkin voidaan toteuttaa SDL-kirjastolla ja samalla mahdollistaa tiedostojen nosto ja pudotus suoraan ohjelman ikkunaan (Drag & Drop). Monen tiedoston yhtäaikainen pudotus on mahdollista, mutta tätä ominaisuutta ei testattu kaikilla ohjelman tukemilla käyttöjärjestelmillä ja alustoilla. Prototyyppi ei tue tulostimia tai skannereita.

3. Projektitiedostot

K-DI-editori kykenee muodostamaan tiedostoista ja hakemistoista kokonaisuuksia, joita kutsutaan projekteiksi. Tämä toiminnallisuus on toteutettu useissa ohjelmointiympäristöissä projektitiedostoilla, jotka sisältävät informaation eri tiedostojen odotetusta sisällöstä ja niiden toiminnallisuudesta.

Projektitiedostoissa voivat olla esitettynä myös eri luokkien väliset suhteet tai jopa vuokaavio koko ohjelman toiminnasta. Prototyyppiä varten projektitiedostossa ei toteutettu kaikkia määriteltyjä toiminnallisuuksia. Vaan vain keskeisimmät, kuten lähdekooditiedostot, kirjastot, projektitiedostot (esimerkkikoodi 1), objektit ja käännösskripti. Eri tietorakenteita määrittävät käskyt ovat [Classes], [Structures], [Properties], [Theme], [Components] ja [KeyBindings].

```
[Properties]
projectName = “My Project”
path = “.”
linkLibraries = “-lSDL -lSDLttf -lSDLmain”
cflags = “-O3”
company = “”
flags = “PATH_ABSOLUTE CASE_SENSITIVE”

[Theme]
textColor=”#AAAAAA”
functionColor = “#BBAAAA”

[Components]
files = “makefile main.cpp myClass.cpp”
headers = “main.h myClass.h”
```

Esimerkkikoodi 1. K-DI-editorin projektitiedosto.

Luokkahierarkianäkymää varten tiedostomuoto sisältää myös täysin abstrakteja elementtejä, joilla ei ole ilmentymää tiedostoselaimessa. Niitä ovat luokkamääritelmä ProjectClass, luokkamääritelmän elementti ProjectClassElement ja elementtien välistä suhdetta kuvaava luokka ProjectElementRelation. Elementtien välistä suhdetta kuvaava luokka on suunniteltu kuvaamaan mahdollisimman moninaisia tilanteita.

Projektin elementit toteutettiin ohjelmointiteknisesti periyttämällä luokkia alaluokkiin. Yläluokkana (esimerkkikoodi 2) on ProjectElement, josta periytyy alaspäin konfiguraatioita, kooditiedostoja, kirjastoja, kansiota, tiedostojen välisiä suhteita jne. Jokaisella tiedostolla ja tekstinpätkällä on myös ”hash” eli koko sen sisällöstä ohjelmallisesti tuotettu lyhyt tunniste, jota voidaan hyödyntää esimerkiksi tiedostojen sisällön samankaltaisuuden vertailussa. IDGenerator-luokan tunnisteenluontifunktio Evaluate() tuottaa samankaltaisesta tekstistä aina samankaltaisen tunnisteen. Tämä ei sovi tietojen salaamiseen, vaan nimenomaan helpottaa erilaisten tekstien kokoamista ja erittelyä. Samaa tunnistemenettelyä on mahdollista soveltaa myös SpellChecker-luokassa, joka on vastuussa oikeinkirjoituksen tarkistamisesta. Tässä yhteydessä väärin kirjoitettu sana linkittyy ohjelmalliseen lähimpään oikeinkirjoitusehdotukseen, mikäli kirjoitetun sanan ja ohjelmallisen arvauksen eroavaisuus on tarpeeksi pieni.

Tunnisteita vertaamalla voidaan selvittää, kuinka samankaltaisia kaksi määräämättömän pituista merkkijonoa ovat. Tunnisteet ovat 32-bittisiä kokonaislukuja eli tyyppiä *int32_t* (esimerkkikoodi 2), joten erilaisia vaihtoehtoja on hieman yli neljä miljardia.

```
#include ”IDGenerator.h”

class ProjectElement {
      protected:
            int32_t _id;
            char *_name = nullptr;
            void *_buffer = nullptr;
            char _flags;
      public:
            virtual void Init() = 0;
            bool isInitialized() { return _flags & IS_INITIALIZED; }
            void generateID() { _id = IDGenerator.Evaluate(buffer); }
            int32_t getID(){ return _id; }
};
```

Esimerkkikoodi 2. Abstrakti yläluokka ProjectElement, josta periytetään muut projektia kuvaavat luokat.

Kehitettävänä olevaa projektia kuvaava luokka sisältää tietueen ProjectProperties (esimerkkikoodi 3), jossa on määriteltynä projektin nimi, käännösparametrit ja muut kehitystyön kannalta keskeiset tiedot. Näistä kahdesta tietorakenteesta voidaan koostaa luokka, joka kuvaa K-DI-ympäristössä kehitettävänä olevaa projektia.

```
#define STR_MAX_LENGTH 1024

struct ProjectProperties {
          char *projectName;
          char *path;
          char *linkLibraries;
          char *cflags;
          char *projectFile;
          char *company;
          char *license;
          time_t lastEditTime, creationTime;
          char memFlags, flags;
};
```

Esimerkkikoodi 3. Projektitiedoston metatiedot, ProjectProperties.

Esimerkkikoodiin on merkitty muuttuja ”flags”, joka sisältää erilaisia ohjelmointiteknisiä lippuja. ProjectProperties määrittelee 16 eri lippua, joista 8 koskee muistin varaamista ja 8 projektin muita ominaisuuksia (taulukot 3 ja 4). Liput on jaettu kahteen muuttujaan, memFlagsiin ja flagsiin. Merkkijonoja kuvataan char-tyypin osoittimina. Vaihtoehtona olisi käyttää C++:n std::string -luokkaa. Osoitin char-taulukkoon valittiin tietotyypiksi nopeutensa ja yksinkertaisuutensa vuoksi.

3. Liput muistin varaamista varten.

Nimi	**Kuvaus**
NAME_RESERVED	Onko muistia varattu nimeä varten?
PATH_RESERVED	Onko muistia varattu hakemistopolkua varten?
LIBRARIES_RESERVED	Onko muistia varattu kirjastoparametreja varten?
CFLAGS_RESERVED	Onko muistia varattu C-käännöslippuja varten?
PROJECT_FILE_RESERVED	Onko projektitiedostoa olemassa?
COMPANY_RESERVED	Onko valmistajan nimeä varten varattu muistia?
LICENSE_RESERVED	Onko lisenssiä varten varattu muistia?
MEMORY_ERROR	Onko muistin varaamisessa ongelmia?

Projektitiedoston syntaksiin kuuluu operaattoreita. Operaattori [] määrittää, mitä parametreja käsitellään. Parametrin arvo asetetaan operaattorilla =. Mikäli jotain arvoa ei aseteta projektitiedostossa, sille asetetaan määrätty oletusarvo. Tiedosto loppuu tyhjään riviin. Kommentit merkitään ”//”-etuliitteellä. Lippu PROJECTFILE_ENABLED (taulukko 4) määrittää, luetaanko projektitiedostoa muistiin.

4. Muut käytetyt ohjelmaliput.

Nimi	**Kuvaus**
PROJECTFILE_ENABLED	Onko projektitiedosto toiminnassa?
PROJECTFILE_AUTOSAVE	Tallennetaanko muutokset automaattisesti?
LIBRARIES_RESERVED	Onko muistia varattu kirjastoparametreja varten?
PATH_ABSOLUTE	Käytetäänkö suhteellista vai absoluuttista tiedostopolkua?
CASE_SENSITIVE	Onko kirjainkoolla merkitystä tiedostojen nimeämisessä?
LINE_ENDING_RETURN	Tuleeko rivinvaihtomerkinjälkeen palautusmerkkiä?
FUTURE_FLAG1	Tulevaisuutta varten varattu lippu.
FUTURE_FLAG2	Tulevaisuutta varten varattu lippu.

Tärkeä huomio on lipussa CASE_SENSITIVE, joka on oleellinen, kun projektitiedostoja siirretään eri käyttöjärjestelmien välillä. Windows-järjestelmä on suunniteltu niin, ettei isoilla ja pienillä kirjaimilla ole muuta kuin esteettistä merkitystä tiedostojen ja kansioiden nimissä. OS X - ja Linux-järjestelmissä taas isot ja pienet kirjaimet merkitsevät eri asiaa, ja on syytä määrittää tämä lippu sen mukaan, missä ympäristössä kehittäjä aikoo projektin kääntää. Suositeltavaa on kirjoittaa kaikki tiedostojen nimet samalla tavalla, joten sekaannuksilta vältytään.

Toinen lippu, PATH_ABSOLUTE, on myös useiden käyttöjärjestelmien välillä liikkuessa huono valinta, koska eri käyttöjärjestelmissä kotihakemistot ovat eri sijainneissa pitkin massamuistia ja absoluuttiseen polkuun navigointi voi olla mahdotonta. Vastakohta tälle ovat relatiiviset hakemistopolut, eli tiedostoja haetaan ali- ja ylähakemistoista täysin tietämättömänä siitä, missä päin tiedostojärjestelmää ollaan.

Kokonaista kehitettävänä olevaa projektia kuvataan luokalla ProjectClass. Se on koosteluokka muista tietorakenteista. Jokainen ProjectClass sisältää ProjectClassProperties-rakenteen ja listan ProjectElement-luokan ilmentymiä. Ilmentymät voivat olla mitä tahansa ProjectElement-luokan alaluokkia, koska itse ProjectElement on abstrakti luokka. Abstrakteista luokista ei voida luoda lainkaan ilmentymiä.

ProjectSource on tietorakenne lähdekooditiedostoille. Siihen on tallennettu lähdekooditiedoston käyttämä ohjelmointikieli. Prototyyppi tukee vain C- ja C++-kieliä. Projektissa mukana oleva CAnalyzer/CPPAnalyzer-luokka käy ohjelmakoodin läpi ja päättelee käytetyn kielen automaattisesti. ProjectSourcessa on lippuja, jotka otetaan huomioon käännösvaiheessa. ProjectSourcen liput määrittelevät käännöksen optimointitason, käytetyn C++-kielen versionumeron ja muita erikoistoiminnallisuuksia.

ProjectHeader poikkeaa ProjectSourcesta siinä, että sillä määritellään C- ja C++-kielien kirjastotiedostoja. Tiedostopäätteet ovat .h ja .hpp. Jokainen ProjectHeader on tarkoitus käydä läpi vain kerran, kun lopullinen ohjelma linkitetään. Tätä varten ohjelmaan on rakennettu valmiiksi aihio (template) näille tiedostoille: jokainen kirjastotiedosto määrittää esikääntäjälle välitettävän muuttujan __KIRJASTO_H__, jossa sana KIRJASTO korvataan kyseisen kirjastotiedoston nimellä. Näin vältetään yksinkertaisella tavalla kirjastojen ristiin linkittäminen.

Kun jokin ProjectSource-elementti toteuttaa ProjectHeader-elementissä määritellyn luokan, järjestelmä vaatii keinon tämän riippuvuussuhteen esittämiseen. Tätä varten on olemassa luokka ProjectElementRelation, jossa on määritetty kaksi ProjectElement-instanssia ja niiden välinen suhde. Elementtien välillä voi olla neljänlaisia suhteita: elementti sisältää määritelmän, elementti sisältää implementaation, elementti määrittelee sisältöä tai elementti määrittelee isäntäilmentymän.

4. Visuaaliset elementit

K-DI-editorin käyttöliittymässä on erilaisia visuaalisia elementtejä. Ne kaikki ovat SDL_Surface -tyyppisiä bittikarttoja. Elementtejä voidaan asetella uudelleen ja luoda uusia, mutta oletusnäkymässä ovat esillä vain tietyt elementit. Oletusnäkymää voidaan

muokata vain lähdekoodista, mutta seuraaviin K-DI:n versioihin ikkunoiden asettelua varten täytyy määritellä tarvittava konfiguraatiotiedosto.

Rakennekaaviogeneraattori lukee halutut konfiguraatio- ja ohjelmakooditiedostot läpi, ja se toteuttaa kaavion grafiikkapuskuriin halutuilla parametreillä ja sisällöllä. Puskurin sisältö voidaan käsitellä ja piirtää ruudulle ohjelman keskussilmukan avulla. Luokkarakennekaaviosta kehittäjä voi helpolla tavalla nähdä visuaalisen esityksen työstämästään ohjelmasta.

Hierarkianäkymässä K-DI suodattaa projektitiedostossa olevien parametrien mukaisesti projektin sisältämät tiedostot ja kansiot listaan, jota voi selata ohjelman käyttöliittymässä. Ulkoasultaan lista on hyvin tavanomainen, kansiota hiiren osoittimella painaessa se avautuu ja sisällä olevat tiedostot paljastuvat. Erilaisia tuettuja tiedostomuotoja ovat jo aikaisemmin luetellut lähdekooditiedostot, kirjastot ja skriptitiedostot. Kaikilla tiedostojärjestelmäelementeillä voi olla metatietoja. Kaksoisnapsautus tiedoston kohdalla avaa valitun tiedoston.

5. Ohjelmakoodin piirtäminen TrueType-kirjasimilla

Ohjelmakoodi on erityislaatuista tekstiä. Yksi merkittävä ominaisuus erottaa sen selkeästi muusta tekstistä: eri kirjaimilla on oltava sama leveys, jotta koodi pysyy luettavana. Ohjelmakoodilla kuvataan usein tilanteita, joissa jotain rakennetta toistetaan tai merkinnät täytyy kirjoittaa tarkasti samalle kohtaa riviä kuin aikaisemmilla riveillä. Tätä varten K-DI-editorissa päädyttiin ratkaisuun, että jokainen kirjasintyyppi pakotetaan monospace-formaattiin. Monospace tarkoittaa, että kaikki merkit ovat samanlevyisiä. Joidenkin kirjasintyyppien kanssa pakotettu skaalaus aiheuttaa vääristymiä, mutta nämä vääristymät eivät heikennä luettavuutta merkittävästi.

6. Muistivuodot

Kuten luvussa 5.3 on jo mainittu, muistin käsittely C++-kielessä tunnetusti vaatii suurta tarkkuutta, joten myös se tuottaa todella paljon ongelmatilanteita. Muistivuotojen paikkaamiseen ja paikantamiseen käytettiin Linux-maailman ohjelmistoa Valgrind, joka ajaa ohjelman läpi ja pitää kirjaa keskusmuistiin tehdyistä varauksista ja vapautuksista.

Kuvassa 6 on esimerkki K-DI:n kehitysversion ajosta Valgrindin läpi. Kuvasta 5 käy ilmi, että K-DI varaa ilman avonaisia tiedostoja 1,1 megatavua keskusmuistia.

Kuva 6. Muistivuotojen tutkimista Valgrindin avulla.

Havainnekuvasta (kuva 5) käy ilmi, että ohjelmasta vuotaa muistia, vaikka pahimmat ongelmat on selvitetty. Muistinvarauksia (allokaatioita) on enemmän kuin muistinvapautuksia (frees/deletes). 153 tavua menee varmasti hukkaan, ja epäsuorasti menetetään 1728 tavua. Toistaiseksi on jäänyt selvittämättä, mistä tämä johtuu, koska tämä virhe ei kaada ohjelmaa. Vakaustestissä K-DI pysyi päällä 3 vuorokautta ilman minkäänlaisia ongelmia. Pahojen muistivuototapausten vuoksi ohjelmakoodia on jouduttiin paikoin kirjoittamaan kokonaan uudestaan.

Usean tiedoston yhtäaikainen pudottamalla vienti ohjelmaikkunaan tuottaa joissain järjestelmissä virhetilanteita. Tämä on SDL:n rajoite. K-DI on tarkoitettu olemaan täysin käyttöjärjestelmistä riippumaton, joten tätä toiminnallisuutta ei voida taata jokaisessa käyttötapauksessa. Mikäli käyttöjärjestelmä tukee tätä toiminnallisuutta, kaikkien valittujen tiedostojen nimet listataan samassa merkkijonossa. Merkkijonossa tiedostojen nimet erotetaan välilyönnillä. Mikäli itse tiedoston nimi sisältää välilyönnin, päädytään vaikeaan tilanteeseen, jossa tulkki ei tiedä, alkaako uuden tiedoston nimen lukeminen vai onko kyseessä aikaisemman tiedoston nimen osa.

7. Koodianalyysi

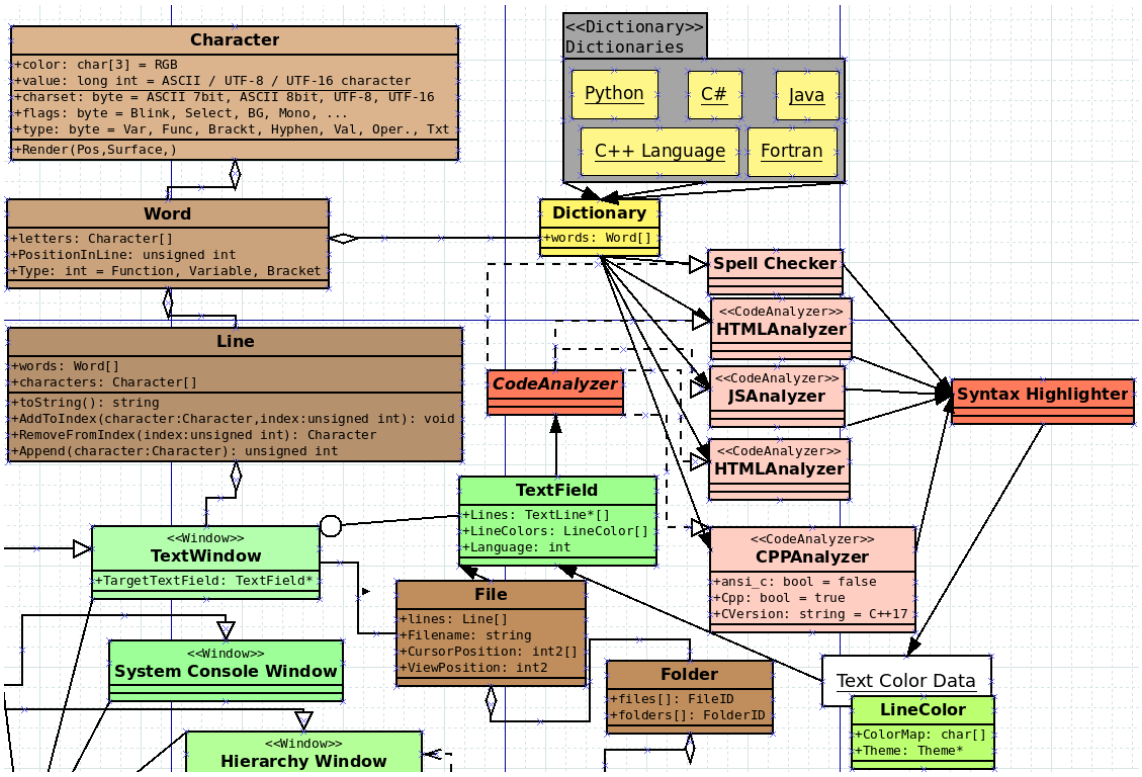
Näytöllä ja keskusmuistissa sijaitsevan koodin analyysiä varten tässä työssä suunniteltiin luokka CodeAnalyzer. Analyysiä tekevä luokka eli analysaattori saa syötteenä ohjelmakoodia ja tuottaa analyysin tuloksen palautusarvonaan. Se on abstrakti luokka, josta periytetään eri ohjelmointikieliä vastaavia aliluokkia [4].

```
class CodeAnalyzer
 {
      protected:
            enum states { NotAnalyzed = 0, UnableToAnalyze = 1, Analyzed = 2 };
            char language;
            char state;
      public:
            void SuperInit(){ language = Unknown; state = NotAnalyzed; }
            virtual AnalyzerConclusion *Evaluate(const char *code) = 0;
 };
```

Esimerkkikoodi 4. Koodianalysaattorin pääluokka.

Jokaisessa *CodeAnalyzer*-luokasta periytyvässä alaluokassa toteutetaan *Evaluate-metodi*, joka on luokan keskeisin toiminnallisuus. Sen palautusarvo, *AnalyzerConclusion,* sisältää analysoitavaan koodiin liitettäviä väriarvoja, funktioiden nimiä, muuttujien nimiä ja eri sulkumerkkien määräämien alueiden sijainnit. Vastinsulkeiden sijainnit tallennetaan muistiin, ja yksinäiset sulkeet ilman paria värjätään virhetilannetta osoittavalla värillä. Kirjanpito näistä alueista mahdollistaa funktioiden, luokkien, parametrilistojen ja muiden yksittäisten kokonaisuuksien erittelyn.

Analysoitavissa olevia ohjelmointikieliä on ohjelmaan suunnitteilla lisää. Tarkoituksena on myöhemmin etsiä kirjasto, joka analysoi koodin syntaksia, ja korvata opinnäytetyön tekemisen aikana syntynyt ohjelmakoodi valmiilla ratkaisulla. Koodin analysointi on hyvin monimutkainen ongelma, eikä sen toteuttaminen onnistu nopeasti. Analysaattori on määritelty luvun 4 luokkakaaviossa. Luokkakaaviosta on tässä luvussa suurennos, jonka avulla analysaattorin toimintaa voidaan hahmottaa selkeästi (kuva 7). Suurennoksessa on esitelty komponentit *Dictionary, CodeAnalyzer* ja *Syntax Highlighter.*



Kuva 7. Koodianalysaattorin luokkarakenne.

8. Kohdistin

Tekstitiedossa liikkuminen on intuitiivisesti hyvin triviaali ongelma. Rivit ovat kuitenkin eri pituisia ja kohdistimen täytyy muistaa, mistä kohdasta tekstiä sitä on liikutettu. Jos kohdistin siirretään riviltä, joka on pidempi kuin alkuperäinen rivi, kohdistimen kohta x-akselilla ei muutu. Mikäli rivi on lyhyempi kuin alkuperäinen rivi, kohdistin siirretään x-akselilla rivin viimeisen merkin kohdalle. Kohdistimen muistiin merkitään, millä rivillä se on aikaisemmin ollut. Näin estetään kohdistimen siirtyminen väärään kohtaan, mikäli sitä siirretään riviltä toiselle. Sijaintihistorian pituuden täytyy olla riittävän suuri, jotta tekstinkäsittely olisi sujuvaa.

Monissa tekstieditoreissa on tuki usealle kursorille samaan aikaan, jotta ohjelmakoodia voi kirjoittaa tehokkaammin. Usean kursorin tukea ei toteutettu K-DI:ssa, vaikka itse konsepti on hyvä ja toisi lisäarvoa editorille. Monen kursorin yhtäaikainen siirtäminen on haastava looginen ongelma.

9. LibPNG-kirjasto

Projektia tehdessä kohdattiin odottamaton ongelma, jossa LibPNG:n kehitysversio ei toimi oikein SDL2:n TrueType-fonttikirjaston kanssa. Tuore Ubuntu Linux -asennus ei kyennyt kääntämään ohjelmaa, mutta toisaalta tuore Debian Linux pystyi siihen. Joissain Ubuntu-asennuksissa käännös ei testikoneilla onnistunut, vaikka kirjastosta oli 64- ja 32-bittiset versiot käytössä. Mikäli järjestelmä ei tue TrueTypeä, editorissa on myös kovakoodattu perusfontti, jota voidaan soveltaa eri käyttötilanteisiin, mikäli haluttua fonttia ei voitu ladata. Mahdollisesti jakeluversiota varten K-DI:n projektikansioon voisi pakata kaikki yhteensopivat kirjastot ja ne käännettäisiin samalla kuin itse projekti. Yksi vaihtoehto on linkittää kirjastot staattisesti jakeluversioon, jotta yhteensopivuusongelmilta vältytään tulevaisuudessa.

6. Pohdinta ja jakelu

Lopputuloksena insinöörityössä tuotettu koodieditorin prototyyppi ei ole täysin toimintakuntoinen. Toimivalta editorilta vaaditaan kyky muokata tekstiä intuitiivisesti ja hallita tiedostojärjestelmän sisältöä käyttöliittymän välityksellä. Käyttöliittymäkomponenteista jäi opinnäytetyön aikana suurin osa vielä toteuttamatta. Karun käyttöliittymän takana on kuitenkin monia toiminnallisuuksia, joita hallitaan toistaiseksi vain muokkaamalla ohjelman lähdekoodia.

Omatekoisten työkalujen käyttäminen ohjelmistojen kehitystyössä on haastava tavoite. Koodieditorin tuottaminen on oleellinen askel. Koodieditori on ohjelmoijan tärkein työkalu kääntäjän ohella. Omaa kääntäjää K-DI:iin ei ole suunnitteilla, vaan se kytkeytyy pelkästään GNU-ympäristöön. Tulevaisuudessa K-DI saattaa tukea versionhallintajärjestelmiä ja useaa yhtäaikaista kehittäjää samassa projektissa verkon välityksellä.

Editorista on mahdollista tehdä myös selainpohjainen käännös. Ruotsalainen projekti nimeltä EMScripten mahdollistaa C- ja C++ -koodin kääntämisen WebASM-muotoon. Tämä voisi avata mahdollisuuden monelle käyttäjälle käyttää ja kokeilla editoria pienemällä kynnyksellä. EMScripten tukee SDL-kirjastoa ja siten olisi suoraan toiminnallinen tämän ohjelmiston kanssa. EMScripten tulee projektiin mukaan mahdollisesti sen jälkeen, kun K-DI hyväksytään opinnäytetyönä.

Käyttöarvo on suuri niissä paikoissa maailmaa, missä Microsoftin tuoteperheeseen ei pääse käsiksi tai sitä ei syystä tai toisesta voi käyttää. Tuote ei kilpaile Microsoftin Visual Coden kanssa, vaan se tarjoaa samanlaisia ominaisuuksia ja mentaliteettia pienemmällä vaivalla ja ilman sitoutumispakkoa lisenssiehtoihin. K-DI on täysin vapaasti käytettävissä, eikä se vaadi sitoutumista sopimuksiin. Telemetriatietoja ei myöskään kerätä. Käyttöarvoa laskee UTF-merkistön tukemattomuus, joten ainoaksi vaihtoehdoksi jää käyttää latinalaisia merkkejä.

Ylläpidettävyyden kannalta ohjelmakoodin taso on hyvä, koska eri toiminnallisuudet on selkeästi jaettu eri lähdekooditiedostoihin. Tiedostoja ei ole dokumentoitu, eikä kommentteja ole kuin monimutkaisimmissa paikoissa. Ohjelmakoodin tulisi olla

itseselitteistä, ja kommentteja tulisi välttää aina, kun mahdollista. Yksi ongelma on rivitys, jonka tulisi olla eri käyttöjärjestelmien kanssa täysin yhteensopiva. Vasta kun tiedosto tallennetaan, eli renderöidään massamuistiin, rivinvaihdot muutetaan projektitiedostossa määritettyyn muotoon. Mikäli projektitiedostoa ei ole, käytetään rivinvaihtona pelkkää rivinvaihtomerkkiä, ”\n”. Vanhemmissa järjestelmissä tämä voi tuottaa ongelmatilanteita.

Mikäli taloudellinen monetisaatio tulee kyseeseen, ohjelmiston Pro-lisenssin hinnaksi on suunniteltu 50 euroa. Monetisaatiota suunniteltaessa täytyy verrata omaa tuotettaan jo markkinoilla oleviin tuotteisiin ja kyetä tarjoamaan jotain sellaista, mitä markkinoilla ei vielä ole. Liian halpa hinta saattaa karkottaa osan asiakkaista, koska se voi kieliä huonolaatuisesta tuotteesta. Monetisaatiossa tulee ottaa huomioon myös lisenssiseikat. Kaikkia kirjastoja ei välttämättä saa käyttää kaupallisissa jakeluissa. K-DI:iin voisi myydä myös mahdollisesti kuukausimaksulla toimivaa Online-lisenssiä.

Projektin etenemistä voi seurata Internetissä osoitteesta http://k-di.eu. Jatkokehityksessä tulee ottaa huomioon kaikki käyttäjät, joten tuettujen alustojen määrä kasvaa entisestään. Tavoitteena on luoda itsenäinen kehitysympäristö, jolla on miellyttävää luoda uusia projekteja. Mikäli K-DI:n ohjelmakoodi pysyy avoimena tulevaisuudessakin, sitä tarjotaan mukaan myös suosituimpiin Linux-jakeluihin.

7. Yhteenveto ja tulokset

Opinnäytetyössä tuotettiin mahdollisimman yksinkertainen teksti- ja koodieditori. Projekti sujui kauttaaltaan hyvin, ja suurilta ongelmilta vältyttiin. Linux osoittautui vaikeaksi käyttää, eikä käännöstä saatu tehtyä jokaisella alustalla loppuun asti. Ohjelma täyttää sille luvussa 4 osoitetut vaatimukset, vaikkakin joitain ratkaisemattomia ongelmia jäi.

Välilehtien toiminallisuutta ei aikarajan puitteissa voitu toteuttaa, joten ohjelma toimii vain yhdellä välilehdellä ja yhdellä avonaisella tiedostolla kerrallaan, vaikka tietorakenteet on suunniteltu tukemaan useita samanaikaisia tiedostoja. Toteutuneista ominaisuuksista laadittiin lista (taulukko 5) vastaamaan luvussa 4 olevaa listaa ominaisuusmäärittelystä.

5. K-DI-editorin ominaisuudet 13.4.2020.

Ominaisuus	**Toteutuksen tila**
Elementtien piirto ruutuun	Kiitettävä
Näppäimistöohjaus	Hyvä
Dynaaminen käyttöjärjestelmäikkuna	Kiitettävä
Lähdekoodin luku ja kirjoitus	Välttävä
Kohdistin	Välttävä
Tunnistegeneraattori	Hyvä
Projektitiedoston luku ja kirjoitus	Välttävä
Ohjelmakoodin analyysi ja värjäys	Heikko
Tiedostojen pudotus ikkunaan	Hyvä
Konsolinäkymä	Heikko
Luokkanäkymä	Toteutus puuttuu!
Hierarkianäkymä	Heikko
Vapaa näppäimistökartta	Hyvä

K-DI:n prototyyppi ei missään tapauksessa ole kaupallinen tuote, joten se voi toimia esimerkiksi lähtöalustana jollekulle muulle ohjelmoijalle toteuttaa markkinoille uusi tekstieditori. Lähdekoodi on kirjoitettu mahdollisimman yksinkertaiseksi ja helpoksi lukea. Käytettyjä ulkopuolisia kirjastoja on todella vähän, ja kääntäminen suoritetaan Makella.

Lähteet

Market share of the most used C/C++ IDEs in 2018, statistics and estimates. 2018. Verkkoaineisto. Coppola, Davide. <http://blog.davidecoppola.com/2018/02/market-share-most-used-c-cpp-ides-in-2018-statistics-estimates/>. 2.2.2018. Luettu 12.3.2020.

Editor War. Verkkoaineisto. Wikipedia. <http://en.wikipedia.org/Editor_War>. Luettu 1.3.2020.

Turbo C++. 2019. Verkkoaineisto. H&V Media Limited. <https://www.filehorse.com/download-turbo-c/>. 14.6.2019. Luettu 20.3.2020.

SDL2 Wiki. 2020. Verkkoaineisto. <https://wiki.libsdl.org/>. Luettu 1.2.2020.

Valgrind Home. 2020. Verkkoaineisto. Valgrind Development Team. <https://valgrind.org/>. Luettu 1.2.2020.

QuickBasic 4.5. 2019. Verkkoaineisto. WinWorld. <https://winworldpc.com/product/quickbasic/45>. Luettu 3.5.2018.

Google Test. 2020. Verkkoaineisto. Google. <https://github.com/google/googletest>. Luettu 13.4.2020.

Teach Yourself C++ in 21 Days. 1997. Liberty, Jesse

Programming a Text Editor in C. 2015. Verkkoaineisto. Coimbre, Adam. <https://www.youtube.com/watch?v=TrR-suFO4to>. 7.6.2015. Tutkittu 1.1.2020