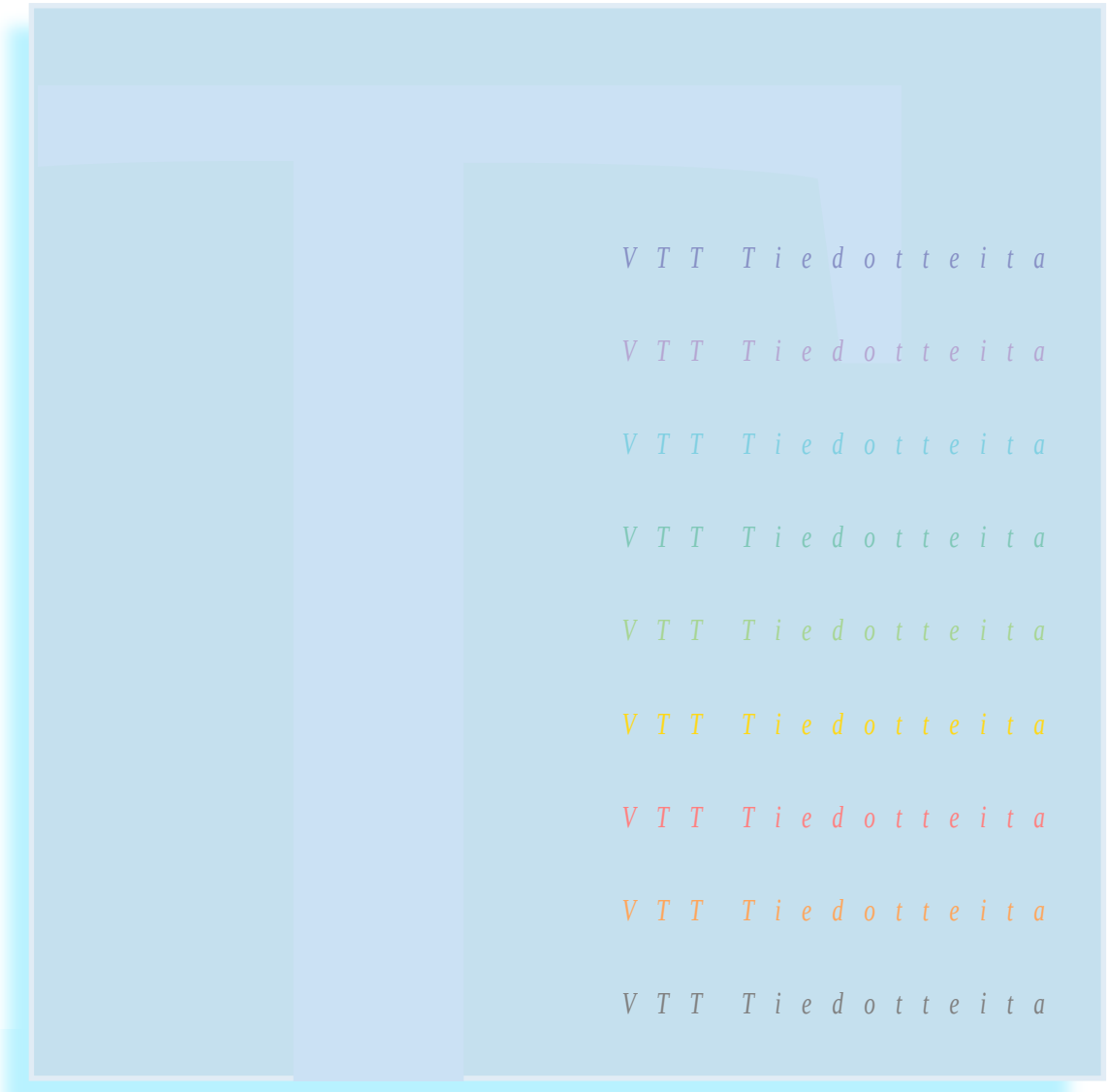
2046

VTT TIEDOTTEITA

Vesa-Pekka Savikko

EPOC-sovellusten rakentaminen



VALTION TEKNILLINEN TUTKIMUSKESKUS ESPOO 2000

VTT TIEDOTTEITA – MEDDELANDEN – RESEARCH NOTES 2046

EPOC-sovellusten rakentaminen

Vesa-Pekka Savikko
VTT Elektroniikka

VALTION TEKNILLINEN TUTKIMUSKESKUS
ESPOO 2000

ISBN 951–38–5688–7 (nid.)
ISSN 1235–0605 (nid.)

ISBN 951–38–5689–5 (URL: http://www.inf.vtt.fi/pdf/)
ISSN 1455–0865 (URL: http://www.inf.vtt.fi/pdf/)

JULKAISIJA – UTGIVARE – PUBLISHER

Valtion teknillinen tutkimuskeskus (VTT), Vuorimiehentie 5, PL 2000, 02044 VTT
puh. vaihde (09) 4561, faksi (09) 456 4374

Statens tekniska forskningscentral (VTT), Bergsmansvägen 5, PB 2000, 02044 VTT
tel. växel (09) 4561, fax (09) 456 4374

Technical Research Centre of Finland (VTT), Vuorimiehentie 5, P.O.Box 2000, FIN–02044 VTT, Finland
phone internat. + 358 9 4561, fax + 358 9 456 4374

VTT Elektroniikka, Sulautetut ohjelmistot, Kaitoväylä 1, PL 1100, 90571 OULU
puh. vaihde (08) 551 2111, faksi (08) 551 2320

VTT Elektronik, Inbyggd programvara, Kaitoväylä 1, PB 1100, 90571 ULEÅBORG
tel. växel (08) 551 2111, fax (08) 551 2320

VTT Electronics, Embedded Software, Kaitoväylä 1, P.O.Box 1100, FIN–90571 OULU, Finland
phone internat. + 358 8 551 2111, fax + 358 8 551 2320

Toimitus Leena Ukskoski

Otamedia Oy, Espoo 2000

Savikko, Vesa-Pekka. EPOC-sovellusten rakentaminen. Espoo 2000, Valtion teknillinen tutkimuskeskus, VTT Tiedotteita – Meddelanden – Research Notes 2046. 56 s. + liitt. 36 s.

Avainsanat EPOC architecture, EPOC applications, graphical interfaces, software development

Tiivistelmä

EPOC on Symbianin kehittämä käyttöjärjestelmä, jonka pääasiallisena kohdealustana ovat kämmentietokoneet, kuten Psion 5mx. Tällaisten laitteiden suosio perustuu pitkälti laajaan ohjelmistotarjontaan: valmiiksi asennetun perussovelluskannan lisäksi tarjolla on laaja valikoima (usein ilmaisia) kolmansien osapuolien tekemiä sovelluksia.

Sovelluskehittäjän kannalta EPOC on tietyllä tavalla kaksijakoinen: toisaalta käyttöjärjestelmä tuntuu hyvin "modernilta", mutta toisaalta jotkut yksinkertaiset asiat ovat turhan hankalia ja etenkin tarjolla oleva dokumentaatio on osin riittämätöntä.

Tämä raportti käy läpi EPOCin arkkitehtuurin yleisellä tasolla ja sen jälkeen käsittelee joitain erityispiirteitä vielä tarkemmin. Asioita tarkastellaan mahdollisimman käytännöllisellä tasolla (mm. koodiesimerkein) ja yksittäisen sovelluskehittäjän näkökulmasta, kritiikkiäkään unohtamatta.

Lopuksi rakennetaan todellinen, julkaistu EPOC-sovellus, Buzz Bingo. Lähdekoodin tarkastelun lisäksi perehdytään myös sovellusten paketointi- ja asennusmekanismeihin.

Sisältö

Lyhenteet

AIF	Application Information File
API	Application Programming Interface
CONE	Control Environment
DLL	Dynamic Link Library
GUI	Graphical User Interface
IDE	Integrated Development Environment
MBM	Multi-Bitmap
MDI	Multiple Document Interface
MVC	Model-View-Controller
RAM	Random Access Memory
ROM	Read Only Memory
SDK	Software Development Kit
UID	Unique Identification

1 Johdanto

Symbian on kulutuselektroniikka- ja tietoliikennejättien omistama yritys, jonka päätuote on EPOC-käyttöjärjestelmä. Kantavana ideana on luoda EPOCista yhteinen nimittäjä osakkaiden eri tuotteiden välille. Tällöin esimerkiksi Nokian matkapuhelin ja Psionin kämmentietokone voisivat sisältää saman käyttöjärjestelmän ja siten jakaa myös ohjelmistotarjonnan. Edellä mainittujen lisäksi Symbianissa ovat mukana Ericsson, Matsushita ja Motorola. Taistelussa seuraavan sukupolven kulutuselektroniikan käyttäjistä Symbian (ja EPOC) muodostaa yhden rintamalinjan. Muita osapuolia ovat mm. Microsoft Windows CE-käyttöjärjestelmineen ja 3Com, jonka Palm-kämmentietokoneet ovat sangen suosittuja. Rajalinjat tosin elävät koko ajan. Esimerkiksi Nokia (ja sitä kautta Symbian) on yhteistyössä 3Comin Palm Computing -tytäryhtiön kanssa ja Ericssonillakin on Windows CE -hankkeita.

Tulevaisuuden markkinaosuuksia ja teknologiavalintoja on luonnollisesti hyvin hankala ennustaa. Tämän selvityksen tarkoituksena on tarkastella EPOCia sovelluskehittäjän näkökulmasta: millä työkaluilla ja kuinka helposti onnistuu EPOC-sovellusten rakentaminen. Valittu näkökulma on siinä mielessä tärkeä, että käyttöjärjestelmä elää ja kaatuu sovellustarjontansa mukana. Jotta EPOC toisi haluttua lisäarvoa kohdelaitteeseensa (esim. matkapuhelimeen), niin perussovellusten lisäksi olisi hyvä olla muutakin tarjontaa ja vieläpä eri lähteistä (*third-party software*).

Aiheen käsittelytapa on ennemminkin katsaus mielenkiintoisiin seikkoihin eikä niinkään kaiken kattava analyysi. Raporttia voidaan tältä osin ajatella eräänlaisena dokumentaationa siitä prosessista, minkä ohjelmoija joutuu käymään läpi siirtyessään EPOC-maailmaan. Aluksi luvussa 2 käydään läpi käyttöjärjestelmän perusrakenne ja sen suunnitteluun vaikuttaneet filosofiat. Tältä pohjalta onkin hyvä siirtyä varsinaiseen ohjelmistokehitykseen luvussa 3, jossa perehdytään C++-kehitysympäristöön ja EPOCin omalaatuisiin piirteisiin, etenkin erilaisten virhetilanteiden hallintaan. EPOC-sovelluksista puhuttaessa ei voida sivuuttaa graafisia käyttöliittymiä, joten niihin liittyviä mekanismeja ja problematiikkaa pohditaan luvussa 4. Raportin konkreettisin ydin on luku 5, jossa todellisen sovelluksen koodin kautta peilataan aiempien lukujen tekniikoita ja lisäksi vielä joitain uusiakin. Loppusanojen (luku 6) jälkeen raportti sisältää liitteenä käsiteltyjen isompien esimerkkien lähdekoodit.

Tarkastelun kohteena oleva EPOCin versio on Release 5. Sitä tukevia laitteita on tällä hetkellä markkinoilla ainakin Psionin ja Ericssonin versiot.

Materiaalia löytyy myös raportin kotisivulta: `http://www.cs.tut.fi/~vespe/epoc/`. Tarjolla on mm. Painoindeksi- ja Buzz Bingo -esimerkkien lähdekoodit, Buzz Bingon asennuspaketti ja muuta asiaan liittyvää.

2 EPOCin rakenne ja filosofia

EPOC sisältää paljon muutakin kuin vain ilmeisimmät käyttöjärjestelmän peruspalvelut. Esimerkiksi yksinkertaisten tallennusformaattien lisäksi EPOC sisältää valmiin relaatiotietokannan, DBMS:n. Myöskin EPOC-laitteen käyttäjälle näkyvät perussovellukset, kuten kalenteri ja taulukkolaskentaohjelma, ovat osa EPOCia. Tämä tarkoittaa sitä, että ohjelmoija voi omissa ohjelmissaan käyttää samoja, sangen kehittyneitä, komponentteja, jotka muodostavat perussovellusten pohjan. Esimerkiksi tekstinkäsittelysovellus käyttää komponenttia, joka tarjoaa suoraan palvelut muotoillun tekstin kirjoittamiseen ja käsittelyyn.

Usein käyttöjärjestelmä typistyy ohjelmoijan näkökulmasta vain enemmän tai vähemmän yksinkertaiseksi APIksi (*Application Programming Interface*). Tällöin oma sovellus on selkeästi kontrolloiva osapuoli, joka ainoastaan tarvitessaan kutsuu jotain käyttöjärjestelmäpalvelua APIn läpi. EPOCin tapauksessa tämä yksinkertainen malli ei enää toimi: käyttöjärjestelmäpalveluita ei voida typistää yksittäiseksi rajapinnaksi, vaan palvelut koostuvat erillisistä, keskenään kommunikoivista yksiköistä. Ohjelmoijalle EPOC tarjoaa sofistikoidun sovelluskehyksen (*framework*): yksinkertaistettuna tämä tarkoittaa sitä, että nyt sovelluskehys kutsuu sovelluskoodia eikä päinvastoin. Kun tähän suoritusmalliin vielä yhdistetään sovelluskehyksen laajuus ja monimutkaisuus tai -puolisuus, niin satunnaisella ohjelmoijalla onkin aika tekeminen kokonaisuuden hahmottamisessa tai edes alkuun pääsemisessä.

Oppimiskynnystä korottaa entisestään laadukkaan dokumentaation (kunnon kirjan) puute. Symbianin kotisivuillaan[1] julkaisemien dokumenttien lisäksi kukin SDK (*Software Development Kit*) sisältää lisäinformaatiota, joka toki tulee tarpeeseen mutta ei välttämättä auta kokonaiskuvan luomisessa. Symbianin koulutusmateriaalikaan [4] ei tunnu olevan niin hyvin jäsennelty kuin toivoisi.

Aloitetaan tutustumalla EPOCin erityispiirteisiin ja sovellusarkkitehtuuriin. Terminologian kannalta lienee syytä mainita, että EPOCin toteutuskieli on C++, joten mainitut palvelut ja luokat löytyvät sinällään EPOC C++ SDK:sta.

2.1 Sovellusten erityispiirteitä

Ohjelmoijan (ja ehkä käyttäjänkin) näkökulmasta EPOC hyödyntää suuressa määrin kahta yleistä ja hyväksi havaittua mekanismia: asiakas-palvelinarkkitehtuuria ja MVC-suunnittelumallia (*Model-View-Controller*). Ensin mainittu tarkoittaa sitä, että esimerkiksi tiedostojärjestelmä on erillisen palvelimen (*File Server*) takana, samoin ikkunointi (*Window Server*). Kommunikointi palvelimen ja sovelluksen välillä tapahtuu erityisten istunto-olioiden avulla. Yksinkertaisempi tapa palveluiden tarjoamiseen olisi perinteinen kirjastoratkaisu. Tämä olisi toki EPOCissakin mahdollista, koska järjestelmä tukee

[1] `http://www.symbian.com`

DLL:iä (*Dynamic Link Library*) mutta asiakas-palvelinmalli tarjoaa seuraavat edut [7]:

- Jaettujen resurssien hallinta helpottuu ja yksinkertaistuu, koska esimerkiksi poissulkemisongelmat ainakin vähenevät elleivät peräti poistu kokonaan, kun resurssien jaon hoitaa yksittäinen palvelinprosessi.
- Palvelin ja asiakkaat ovat kukin erillinen prosessinsa (tai säikeensä alustasta riippuen), jolloin ne eivät voi kovin helposti häiritä (ainakaan vakavasti) toistensa toimintaa.
- Ajastettujen palveluiden toteuttaminen helpottuu, koska palvelin pyörii jatkuvasti, jolloin sovelluksen ei tarvitse jäädä itse odottamaan määräaikaan saakka. EPOC sisältää valmiin ajastinpalvelun (luokka `RTimer` ja kumppanit).

Yleisellä tasolla MVC-malli tarkoittaa nimensä mukaisesti sovelluksen jakamista osiin siten, että eri osat ovat mahdollisimman itsenäisiä ja toisistaan eristettyjä. MVC koostuu seuraavista osista [3]:

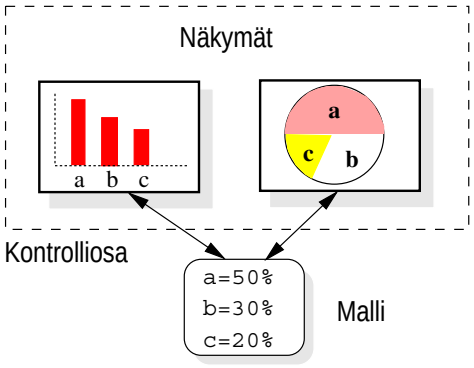
1. Malli (*model*) sisältää varsinaisen sovelluskohtaisen datan.
2. Näkymät (*view*) esittävät mallin sisältämän datan käyttäjälle.
3. Kontrolliosa (*controller*) tarjoaa keinot mallin sisältämän datan manipulointiin. Yleensä tämä tapahtuu näkymien kautta eli graafinen käyttöliittymä sisältää tavallaan sekä näkymät että kontrolliosan, kuten kuvassa 1.

MVC voidaan toki toteuttaa useammallakin tavalla, kunhan mallin ominaispiirteet säilyvät. Esimerkiksi kullakin näkymällä voisi hyvin olla oma kontrolliosansa, joita voitaisiin vieläpä vaihtaa lennossa (*pluggable controllers*) [1].

Tyypillisesti MVC yhdistetään nimenomaan graafisiin käyttöliittymiin ja EPOCissa ohjelmoija jopa jossain määrin pakotetaan toimimaan mallin mukaan, kuten myöhemmissä luvuissa tullaan näkemään.

2.2 Sovellusarkkitehtuuri

EPOCin sovellusarkkitehtuuri Apparc (*application architecture*) määrittelee sovelluksen ja dokumentin käsitteet tai tarkemmin niihin liittyvät rajapinnat. Varsinaiseen EPOC-arkkitehtuuriin liittyy toki muitakin osia, kuten graafisiin käyttöliittymiin liittyvät CONE (*Control Environment*) ja EIKON. Käsitellään nyt Apparciin liittyviä seikkoja jonkinasteisella tarkkuudella. Syvempi ymmärrys asiasta saavutettaneen myöhempien lukujen esimerkkien avulla.



Kuva 1. MVC-malli.

Saumattomuus. Sovellusten asentamisen tulisi tapahtua riittävän kivuttomasti. Samoin järjestelmän pitää huomata sovelluskannassa tapahtuneet muutokset. Ensimmäinen tavoite saavutetaan tiedostojärjestelmän yksinkertaisuudella. Tarkemmin sanottuna kullakin sovelluksella on oma, tarkkaan määritelty paikkansa eli sovelluskohtainen hakemisto hakemiston `\system\apps` alla. Nyt ovat kyseessä nimenomaan sellaiset EPOC-sovellukset, jotka näkyvät EPOC-laitteen (tai emulaattorin) Extras-valikossa. Näiden `app`-päätteisten sovellusten lisäksi olisi toki mahdollista tehdä tavallisia kirjastoja tai näkymättömissä pyöriviä palvelimia. Eri ohjelmatyypit luetellaan taulukossa 1. Ohjelmatyyppien tunnisteet määräytyvät `makmake`-työkalun mukaan. Työkaluun palataan C++:n merkeissä myöhemmin.

Saatavilla olevat sovellukset löydetään `CApaAppFinder`-instanssin avulla (tai tarkemmin luokasta periytetyllä `CApaScanningAppFinder`-instanssilla). Sovelluksia etsitään kunkin näkyvissä olevan levyn `\system\apps`-hakemistosta. Kannattaa muistaa, mitä termi levy esim. Psionin tapauksessa tarkoittaa: levy `Z` on laitteen ROM-muistia, (ainakin) levy `C` RAM-muistia ja lisäksi mahdolliset ulkoiset tallennusvälineet kuvautuvat kukin omaksi levykseen. Löydetyt sovellukset kuvataan `CApaAppData`-instanssien avulla, jotka luodaan kutakin sovellusta vastaavan AIF-tiedoston (*Application Information File*) pohjalta. Tässä vaiheessa sovelluksesta tiedetään seuraavat seikat:

- sijainti
- UID:t (ks. alla).
- sovellusta symboloivat kuvakkeet tai vaihtoehtoisesti oletuskuvake (kysymysmerkki), jos sovellukselle ei ole luotu omia kuvakkeita
- käyttäjälle näkyvä sovelluksen nimi
- järjestelmän kannalta tärkeät piirteet (*capabilities*): tukeeko sovellus upotusta tai uuden tiedoston luontia, onko sovellus näkymätön vai käyttäjän valittavissa Extras-valikosta

Dokumenttilähtöisyys. EPOCissa kuhunkin sovellukseen liittyy yleensä dokumentin käsite. Dokumentti kuvaa sovelluksen kerrallaan käsittelemää dataa. Intuitiivisin esimerkki lienee tekstinkäsittelysovelluksen dokumentti tai

Taulukko 1. EPOC-ohjelmatyypit [7].

exe	Tavallinen, ”perinteinen” ohjelma.
app	Extras-valikossa näkyvä sovellus, jolla on graafinen käyttöliittymä.
dll	Kirjasto, joka ainoastaan tarjoaa muiden sovellusten kaipaamia palveluita.
exedll	Ohjelma, joka monisäikeisessä ympäristössä (varsinaisessa EPOC-laitteessa) on tyyppiä **exe** mutta yksisäikeisessä (kuten Windowsin EPOC-emulaattorissa) **dll**.

piirrosohjelman piirros. Yhtymäkohta MVC:n malliin on ilmeinen. Saumattomuus liittyy myös dokumentteihin siinä mielessä, että EPOC mahdollistaa dokumenttien upottamisen. Esimerkiksi tekstinkäsittelyssä dokumenttiin voidaan upottaa piirrosohjelmalla tehty piirros ja vieläpä siten, että piirrosta voidaan piirrosohjelman avulla muokata tekstinkäsittelyohjelman ”sisällä”. Vastaavantyyppinen tekniikkahan on tuttu myös työpöytämaailmasta (esim. Windows). Mekanismi ei ole tyystin automaattinen, vaan upotettavaa dokumenttia vastaavan sovelluksen täytyy tukea upottamista. Järjestelmä tietää kunakin ajanhetkenä, mitä mahdollisesti upotettavia sovelluksia on saatavilla (ks. yllä).

EPOCin mielestä sovellus ei voi käsitellä kuin yhtä dokumenttia kerrallaan. Merkille pantavaa on, että dokumentin käsite annetaan sovelluskehyksen puolesta valmiina. Toisin sanoen sovellukselta voidaan pyytää sitä vastaavaa dokumenttia, koska kyseinen pyyntömekanismi on aina olemassa (tarkemmin sanottuna `CApaApplication`-luokan `CreateDocumentL`-jäsenfunktio).

Dokumentteihin liittyy kiinteästi käsite UID (*Unique Identification*). Se on yksilöllinen tunniste, joka identifioi sovelluksen. UID:n pääasiallinen käyttötarkoitus on toimia eräänlaisena linkkinä sovelluksen ja sen ymmärtämän datan välillä: kun käyttäjä yrittää avata tiedoston (dokumentin), niin käynnistyykin tiedoston luonut sovellus. Tiedostoon on siis aina talletettu sovelluksen UID. Toisin sanoen oikea sovellus ”tiedetään” eikä sitä tarvitse arvata esimerkiksi tiedostopäätteen perusteella. Yhden UID:n sijaan sovellukseen liittyy kolme UID:tä eli ns. yhdistelmä-UID (*type UID*), joka kuvataan luokalla `TUidType`. Näistä ainoastaan UID3 on ohjelmoijan valittavissa, muut kaksi määräytyvät käyttötarkoituksen mukaan.

Sovelluskehys. Sovelluskehitys tapahtuu EPOCin sovelluskehyksen ehdoilla. Ohjelmoijan kannalta tämä tarkoittaa yksinkertaistettuna sitä, että autonomisten ajettavien ohjelmien sijaan koodataankin DLL:iä ja vieläpä siten, että ne tarjoavat mallinmukaisen rajapinnan.

3 C++

Oletettavaa on, että pääasiallinen EPOC-sovellusten kehitystyö tehdään C++:lla. Yksinkertainen perustelu asiantilalle on se, että EPOC on itsekin toteutettu C++:lla, joten siinä mielessä C++ lienee parhaiten tuettu. Toisaalta tämä tarkoittaa myös sitä, että vähänkään syvällisemmän käsityksen saaminen EPOCin muodostavien sovelluskehysten toiminnasta ei kovin helposti onnistu ilman C++:aa, koska luonnollisesti yksityiskohtaiset kuvaukset päätyvät aina loppujen lopuksi rajapintatasolle eli käytännössä C++ -luokkiin.

Olisi kuitenkin liian yksioikoista väittää, että C++ olisi aina paras mahdollinen tai edes soveltuva valinta. Yksinkertaisen sovelluksen rakentaminen onnistuu varmasti helpoiten OPL:llä tai Javalla. Toisaalta tällainen sovellus ei ole EPOCin ”standardisovellus” siinä mielessä, että se ei kytkeydy sovelluskehykseen niin saumattomasti kuin voisi odottaa. Se, kuinka suuri haitta tuo käytännössä on, riippuu tietenkin tapauksesta. Sekä OPL että Java tukevat myös lähestymistapaa, jossa osa sovelluksesta koodataan C++:lla.

3.1 Kehitysympäristö

EPOC Worldistä[1] ilmaiseksi ladattava EPOC C++ SDK sisältää tarvittavien otsikkotiedostojen ja kirjastojen lisäksi joukon työkaluja, kuten emulaattorin. SDK olettaa, että kehitysympäristönä on Visual C++. Jotta sovellukset saataisiin siirrettyä EPOC-laitteeseen (jossa on ARM-prosessori), sisältää SDK GNU C++-kääntäjän (`gcc`) ja muita työkaluja.

Eri käännösprosessien (Visual C++ vs. `gcc` eli emulaattori vai ei) yhteinen nimittäjä on `makmake`-työkalu, joka generoi annetusta `mmp`-tiedostosta halutunlaisen makefilen. Esimerkiksi Visual C++:n tapauksessa makefilen formaatti olisi sama kuin työkalun ns. workspace-tiedostoilla. Kuten esimerkistä 1 nähdään, niin `mmp`-tiedostossa kerrotaan projektin oleellisimmat piirteet, kuten UID, lähdekooditiedostot ja niiden sijainti, kohdebinääri ja tarvittavat kirjastot.

Esimerkistä 1 voidaan päätellä kaikenlaista: ollaan selvästikin rakentamassa ns. EIKON-sovellusta eli graafisen käyttöliittymän sisältävää ohjelmaa (tyyppi `app`, UID:n ensimmäinen osa eli UID2 on arvoltaan `KUidApp`). UID:stä nähdään lisäksi, että kyseessä on kehitysversio, koska jälkimmäinen UID:n osa (UID3) on välillä 0x01000000 – 0x0fffffff. Välin ulkopuolella olevat ”lopulliset” UID:t täytyy erikseen pyytää Symbianilta päällekkäisyyksien välttämiseksi. UID:n lisäksi sovellukselle voitaisiin määritellä erillinen UNICODEUID, jota käytettäisiin silloin, kun sovelluksestä käännetään Unicode-merkistöä käyttävä versio. Vaikka tällainen mahdollisuus on periaatteessa olemassa, niin kehitysympäristö ei toistaiseksi tue Unicodea.

[1] `http://www.epocworld.com/`

Esimerkki 1. mmp-tiedoston sisältö.

```
TARGET          fats.app
TARGETTYPE      app
UID             0x1000006c 0x010000fa
TARGETPATH      \system\apps\fats
PROJECT         vps
SUBPROJECT      fats
SOURCE          fats.cpp fatsappview.cpp
RESOURCE        fats.rss
USERINCLUDE     .
SYSTEMINCLUDE   \epoc32\include
LIBRARY         euser.lib apparc.lib cone.lib eikon.lib
```

Kohdesovellus käännetään hakemistoon `\system\apps\fats` nimellä `fats.app`. Hakemisto viittaa EPOCin tiedostojärjestelmään, kun taas rivit PROJECT ja SUBPROJECT kertovat lähdekooditiedostojen sijainnin kehityskoneella: hakemisto `\vps\fats`. Hakemistossa on ainakin tiedostot `fats.cpp`, `fatsappview.cpp` ja `fats.rss`. Jälkimmäinen on ns. resurssitiedosto, jossa määritellään mm. käyttöliittymävalikot, dialogit ja käytettävät merkkijonot. Käyttämällä resurssitiedostoja kurinalaisesti ja johdonmukaisesti päästään tilanteeseen, jossa sovelluksen ulkoasua voidaan muuttaa helposti ja kivuttomasti, varsinaiseen lähdekoodiin ja siten sovelluksen toiminnallisuuteen kajoamatta. Käytännön esimerkkinä voidaan mainita uuden kieliversion tekeminen. Resurssitiedosto pitää erikseen kääntää SDK:n `rcomp`-ohjelmalla.

Eri INCLUDE-rivit toimivat niin kuin olettaa sopii: kääntäjän esiprosessori löytää EPOCin systeemiotsikkotiedostot ja käyttäjän työhakemistossa olevat otsikkotiedostot. Polkuja voisi olla rivillä useampiakin välilyönnein eroteltuina. LIBRARY-rivillä kerrotaan mukaan linkitettävät kirjastot. Rivejä voi olla useampiakin.

Visual C++:n valinta käytetyksi IDE:ksi (*Integrated Development Environment*) on jota kuinkin onnistunut[2]. `makmaken` generoima workspace-tiedosto sisältää tarvittavat viittaukset lähdekooditiedostoihin (sekä `mmp`-tiedostossa mainittuihin että `makmaken` itse tekemiin (esim. `fats.uid.cpp`)). Pientä hienosäätöä toki vaaditaan, koska `mmp`-tiedoston tarkoitus on ohjata yksittäinen käännösprosessi kunnialla läpi eikä suinkaan olla makefilen korvike. Tästä syystä `mmp`-tiedosto ei huomioi eri tiedostojen välisiä riippuvuuksia ja otsikkotiedostojahan ei `mmp`-tiedostossa edes luetella.

Kehitystyössä tarvittavat perustyökalut on helppo integroida IDE:n osaksi, esimerkiksi `rcomp` osaksi Tools-valikkoa. Kun halutaan ajaa EIKON-sovellusta (joka itse asiassa on sovelluskehykseen liitettävä DLL), niin IDE osaa ensimmäisellä kerralla kysyä emulaattorin polkua ja sen jälkeen käynnistää sen. Käännettävän

[2] Arviot perustuvat Visual C++:n versioon 6.0.

projektin konfiguraatiokin voidaan valita käännöskohtaisesti: debug tai release. Lisäksi, jos UNICODEUID on `mmp`-tiedostossa annettu, niin konfiguraatiovaihtoehdot sisältävät myös Unicode-variaatiot. Tosin niiden valinta ei paljon mieltä lämmitä, koska vastaava kääntäminen ei kuitenkaan onnistu.

Kehitysympäristön ikävimmät piirteet liittyvät valitun IDE:n ominaisuuksiin ja etenkin editoriin. Tokihan kaikkeen tottuu, mutta etenkin rutinoitunut Emacs-konkari on aluksi ihmeissään Visual C++:n editorin alkeellisuudesta. Niinkin yksinkertaiset asiat kuin sisennysten pakottaminen tai valittujen koodirivien tilapäinen poiskommentointi eivät tunnu onnistuvan millään. Tämä on täysin käsittämätöntä, etenkin kun editori vilisee kertaluokkaa eksoottisempiakin piirteitä (esim. funktion prototyypin näyttäminen pienessä ikkunassa siinä vaiheessa, kun ohjelmoija kirjoittaa ko. funktion määrittelyä). Toki täytyy muistaa, että ärsytyskynnystä tässä tapauksessa alentaa eri kulttuurien — Unix vs. Windows — yhteentörmäys. Toinen vastaava esimerkki on Windowsin MDI-filosofia (*Multiple Document Interface*): sovelluksen (tässä tapauksessa Visual C++:n) kaikki ikkunat ovat yhteisen emoikkunan sisällä. Käytännössä tämä tarkoittaa koodaamisen kannalta sitä, että maksimissaan kaksi lähdekooditiedostoa on yhtä aikaa näkyvissä, useimmiten ainoastaan yksi. Ikkunasta toiseen vaihtaminen on sangen kömpelöä, etenkin jos editoitavia tiedostoja on useampia. Unix-puolella vallitseva filosofia yhdistelmässä X + Emacs on erilainen: kustakin tiedostosta saa halutessaan oman ikkunansa ja, jos eri ikkunat vielä sijoitellaan sopivasti virtuaalityöpöydälle, niin ikkunasta toiseen vaihtaminen sujuu helposti ja eri ikkunat eivät peitä häiritsevästi toisiaan.

Visual C++:n ehkä paras puoli on intuitiivinen ja suhteellisen helppokäyttöinen debuggeri. EPOC-ohjelmoinnissa debuggeri on vieläpä lähes välttämätön, koska vaihtoehtoisten debuggausmenetelmien (esim. välitulostukset) käyttäminen ei etenkään EIKON-sovellusten tapauksessa ole mahdollista (tai ainakaan vaivan arvoista). Ohjelmoija joutuu helposti myös tilanteisiin, joissa dokumentaation puutteita joudutaan kompensoimaan debuggerin avulla. Esimerkkinä voidaan mainita olioiden väliset omistussuhteet (eli kenen vastuulla on dynaamisesti varattujen muistialueiden vapautus), kun käytetään EIKONin tarjoamia alustakontrolleja (*compound control*). Toisin sanoen täytyy selvittää, tuhoaako alusta siihen liitetyt kontrollit oman tuhoamisensa yhteydessä, vaikkei se olisi kyseisiä kontrolleja itse luonutkaan. Nimenomaan tämäntyyppinen debuggaus onnistuu, koska SDK:n mukana tulee osa EPOCin lähdekoodista (lähinnä graafisiin käyttöliittymiin liittyvien luokkien toteutuksia). Debuggeri myös tietää lähdekoodin sijainnin, joten debuggaus onnistuu ainakin osittain ihan lähdekooditasolla.

3.2 Tyypit

Tyyppisysteemi on kaiken perusta, koska yksittäinen muuttuja on hienojakoisin yksikkö, jonka kanssa ohjelmoija tulee työskentelemään. Vaikka käytetään C++:aa, niin tuttujen tyyppien (`int`, `void*`, `string`, ...) sijaan EPOC tarjoaa omat vastineensa. Tämä lähestymistapa pyrkii toisaalta helpottamaan ristiinkäännöstä ja toisaalta syyt piilevät syvemmällä EPOCin filosofiassa.

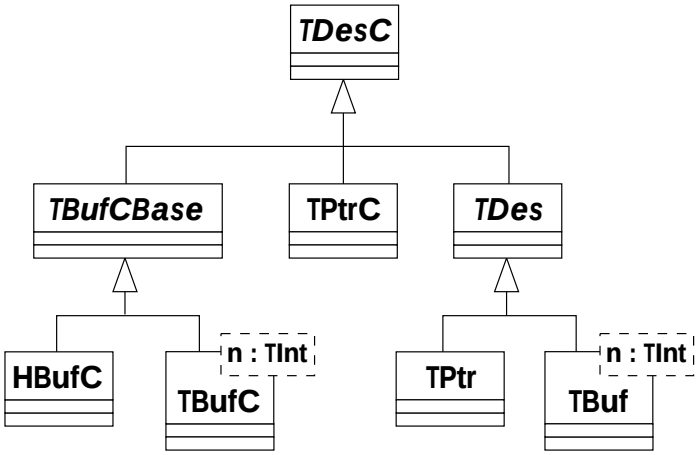
Taulukko 2. Tyyppien nimeämiskäytännöt.

	Selite	**Esimerkki**
T	Sisältää arvon. Luokkana ei omista muita olioita, joten purkajaa ei yleensä tarvitse toteuttaa.	`TInt`, `TDes`, `TAny*`
K	Vakio	`const TInt KMax = 10;`
E	Luettelovakio. Varsinainen `enum`-määrittely on T-tyyppiä.	`ETrue, EFalse`
C	`CBase`-luokan lapsiluokka. Varataan aina dynaamisesti ja ei koskaan välitetä arvona. Ainoastaan C-luokat voivat omistaa dynaamisesti varattuja jäseniä.	`CSession`, `CEikLabel`
R	Toimii välittäjänä (*proxy*) jonkun resurssin ja muun ohjelmakoodin välillä.	`RFile`, `RProcess`
M	Mixin-luokka eli abstrakti luokka, joka sisältää ainoastaan rajapinnan. Yhdessä moniperinnän kanssa mahdollistaa mixin-luokkien toteuksen eli tietyn protokollan implementoinnin.	`MCoeControlObserver`

Aluksi muutama sana nimeämiskäytännöstä. Tyyppien nimet alkavat isolla kirjaimella ja varsinaista nimeä edeltää yksikirjaiminen tunniste. Tunnisteet on kuvattu taulukossa 2. Kaikille EPOC-luokille ei ”virallista” tunnistetta löydy. Esimerkiksi luokka `User`, joka ei sisällä lainkaan dataa, vaan ainoastaan yleishyödyllisiä luokkafunktioita. Joidenkin tyyppien perässä oleva tunniste C tarkoittaa, että olion sisältämää dataa ei voida muuttaa luomisen jälkeen (esim. `TDesC`). Tyyppinimien lisäksi EPOC ulottaa omanlaisensa nimeämiskäytännön myös muihin nimiin (muuttujat, funktiot, …). Koko ohjeiston läpikäynti ei liene tarkoituksenmukaista, joten kyseisiin käytäntöihin tutustutaan lähinnä esimerkkien voimin. Poikkeuksena virhetilanteisiin ja muistinhallintaan liittyvät konventiot, jotka selvitetään asianomaisessa yhteydessä.

Luokassa `CBase` on pari merkillepantavaa seikkaa. Ensinnäkin sille on toteutettu sekä virtuaalinen purkaja että `new`-operaattori. Toisekseen `new`-operaattori on toteutettu siten, että se alustaa varatun muistin nollilla, jolloin varattuun alueeseen instantioidun olion jäsenmuuttujat ovat myös alussa nollattu. Virtuaalinen purkaja pitää luonnollisesti huolen siitä, että kaikissa tapauksissa, missä `CBase`-luokan lapsiluokan instanssi tuhotaan `delete`llä, kutsutaan oikeaa purkajaa.

Taulukossa 2 mainittu mixin-moniperintä on EPOCissa ainoa sallittu (tai ainakin toivottava) moniperinnän muoto. Toisin sanoen lapsiluokka perii toteutuskoodia aina ainoastaan yhdestä lähteestä. Muut kantaluokat määrittelevät palveluita, jotka lapsiluokan täytyy toteuttaa. EPOCin terminologiassa puhutaan protokollasta (*protocol*). Kukin mixin-luokka määrittelee protokollan, joka koostuu mixinin kuvaamasta rajapinnasta. Pohjimmiltaan `CBase`-luokasta ja mixin-luokasta periytet-



Kuva 2. Deskriptorihierarkia

ty lapsiluokka on protokollan tarjoaja (*protocol provider*). Mixin-mekanismin idea on siinä, että sen avulla saavutetaan moniperinnän tärkein hyöty eli eri luokkien joustava tyyppi- tai protokollayhteensopivuus. Kuitenkin vältytään moniperinnän pahimmalta ongelmalta eli toteutustason riippuvuuksilta eri kantaluokkien välillä. Toisin sanoen mixin-luokat ennemminkin jäsentävät luokkahierarkiaa eivätkä suinkaan sekoita sitä entisestään. Yksittäisen luokan esittelystä nähdään suoraan, mitkä mixinit se on sitoutunut toteuttamaan eli missä kaikissa ”rooleissa” luokan instanssit voivat elinaikanaan esiintyä.

Merkkijonot on EPOCissa toteutettu ns. deskriptoreiden (*descriptor*) avulla. Deskriptorit eivät tosin ole rajoittuneet ainoastaan merkkijonoihin, vaan niiden avulla voidaan käsitellä sekä tekstimuotoista että binääristä dataa yhdenmukaisella tavalla. Toisin sanoen deskriptori pitää huolen datan tarvitseman tilan varaamisesta (joko pinosta tai dynaamisesti, deskriptorin tyypistä riippuen) ja tarjoaa turvallisen rajapinnan datan manipulointiin. Turvallisuudella tarkoitetaan tässä sitä, että ohjelmoija ei pysty vahingossa korruptoimaan muistia esimerkiksi indeksoimalla merkkijonon yli.

Eri tarkoituksia varten löytyy erilaisia deskriptoreita. Kaikki deskriptorityypit tosin kuuluvat kuvan 2 mukaiseen hierarkiaan. Hierarkian abstrakti kantaluokka `TDesC` toimii eri deskriptorityyppien yhteisenä nimittäjänä. Koska `TDesC`-instanssia ei toisaalta voida suoraan luoda eikä myöskään luomisen jälkeen muokata, on se käyttökelpoinen esimerkiksi välitettäessä lapsideskriptoriluokkien instansseja funktioille. `TDesC`:n rajapinta sisältää kattavat palvelut merkkijonoa muokkaamattomaan manipulointiin, esimerkkinä indeksointi, vertailut, alimerkkijonojen luonti, haut, jne. Kuten nimi (ei C:tä lopussa) antaa olettaa, `TDes` on muokattava `TDesC` eli se sisältää jäsenfunktiot deskriptorin esittämän merkkijonon (datan) muuttamiselle.

Varsinaiset instantioitavat deskriptorit eroavat toisistaan tilanvarausmekanismiensa suhteen. `TPtrC` ja `TBufC` (vastaavasti tietysti myös `TPtr` ja `TBuf`) varaavat käyttämänsä muistin staattisesti. Näiden deskriptorien ero on siinä, että nimensä

mukaan TPtrC (*pointer descriptor*) ainoastaan osoittaa muualla varattuun dataan, kun taas TBufC:ssä (*buffer descriptor*) data on kiinteä osa deskriptoria. TPtrC on käyttökelpoinen esimerkiksi silloin, kun halutaan käsitellä ROM-muistissa olevaa dataa. TPtrC-instanssin voi alustaa osoittamaan myös johonkin toisen tyyppisen deskriptorin dataan (esim. TBuf) tai C:stä tuttuun merkkijonoesitykseen, joka EPOCissa on tyyppiä TText*. Yleisenä periaatteena onkin, että EPOCin kanssa tulisi aina käyttää deskriptoreita C-merkkijonojen sijaan. Vanhaa teknologiaa ei tosin voida aina loppumattomiin vältellä, joten on toki hyvä, että muunnosmahdollisuudet ovat olemassa puolin ja toisin.

TBufC-varaa pinosta halutun määrän tilaa merkkijonolle. Luokka on toteutukseltaan template, jolle on määritelty erikoistus TInt-tyypille, jolloin instantioimisen syntaksi on hyvinkin intuitiivinen (esim. TBuf<20> buf). Lienee syytä mainita, että TBufC ei nimestään huolimatta ole kaikissa olosuhteissa muuttumaton. Luomalla TBufC-instanssiin osoittava TPtr-olio, päästään kyseisen osoitinolion avulla manipuloimaan TBufC:n sisältämää dataa.

Muistinhallinnan kannalta TPtrC- ja TBufC-luokat johdannaisineen ovat yksinkertaisia ja helposti hallittavia, koska ohjelmoijan ei tarvitse kantaa vastuuta niiden varaamien resurssien vapauttamisesta, sillä kaikki muistinvaraus tapahtuu staattisesti. Toisaalta staattisuus voi joissain tapauksissa olla liian joustamatonta ja kahlitsevaa. Tällaisissa tapauksissa kelvollinen vaihtoehto on HBufC-deskriptori (*heap descriptor*), joka varaa tarvitsemansa tilan dynaamisesti. HBufC:n rajapinta sisältää jäsenfunktioita mm. varatun tilan kasvattamisen ja pienentämiseen. Toki täytyy muistaa, että eri deskriptorityypit pelaavat tarvittaessa ja käytännössä hyvin yhteen: TPtrC (tai TPtr) toimii usein eräänlaisena näkymänä muuntyyppisiin deskriptoreihin ja usein deskriptoria kuin deskriptoria käsitellään TDesC-tyyppisenä. Kuvassa 2 esiintyvä luokka TBufCBase on ainoastaan sovelluskehyksen implementointiyksityiskohta eikä siten käytännön ohjelmoinnissa tarpeellinen.

EPOC-merkkijonoihin liittyy myös Unicode (tai ainakin tulee liittymään tulevaisuudessa, kun tarvittava tuki löytyy sekä kehitys- että ajoympäristöistä). Toisin sanoen yksittäinen merkki vie tilaa joko 8 tai 16 bittiä. Tyyppijärjestelmässä tämä näkyy siten, että eri merkkityypeistä on olemassa useampi versio. Esimerkiksi yksittäistä merkkiä vastaavat tyypit TText8 (8-bittinen merkki) TText16 (Unicode) ja TText. Viimeisen vaihtoehdon koko päätetään käännösaikana, joten 8-bittisestä koodista saadaan Unicode-versio vain käännösoptioita muuttamalla. Mikäli deskriptori sisältää binääristä dataa, niin silloin kannattaa aina käyttää deskriptorin 8-bittistä varianttia (esim. TPtrC8), jotta mahdolliset myöhemmät Unicode-käännökset eivät aiheuttaisi yllättäviä ongelmia. EPOC määrittelee myös seuraavat käyttökelpoiset makrot: _S rakentaa annetusta literaalista oikeanlaisen (Unicode-merkit tai ei) C-tyyppisen vakiomerkkijonon. Makro _L on samantyyppinen mutta nyt literaalista rakennetaan vastaava TPtrC-instanssi.

SDK:n dokumentaation propagandasta huolimatta deskriptorit voivat tuntua aluksi (ja aika kauankin) hankalilta ja oudoilta. Seuraavilla yksinkertaisilla periaatteilla pärjää käytännössä aika pitkälle:

- Pyritään aina käyttämään deskriptoreita eikä taannuta `TText*`-esitysmuotoon.
- Funktioiden argumenttityyppinä `TDesC&` on kevyt (viite), turvallinen (muokkaaminen vahingossa ei onnistu) ja helppo muistaa (minkätyyppinen deskriptori tahansa voidaan välittää).
- Ainoastaan `HBufC`-instanssit luodaan `new`:llä, muut voidaan varata pinosta.
- Eri deskriptorityypit sisältävät muunnosjäsenfunktioita. Esimerkiksi `HBufC::Des` → `TPtr`, `TDesC::Alloc` → `HBufC*`.

Esimerkki 2. Deskriptoriesimerkin `mmp`-tiedosto.

```
TARGET        descex.exe
TARGETTYPE    exe
UID           0x01083941
PROJECT       vps
SUBPROJECT    report
SOURCE        descex.cpp
SYSTEMINCLUDE \epoc32\include
LIBRARY       euser.lib
```

Deskriptorien luonne selkiintynee parhaiten yksinkertaisella esimerkillä. Koska kyseessä on tämän raportin ensimmäinen varsinainen ohjelmaesimerkki, niin se lienee syytä käydä läpi suhteellisen tarkasti. Aluksi muutama sana projektin `mmp`-tiedostosta (esimerkki 2): rakennettava sovellus on exe-tyyppinen ja sillä on tekstikäyttöliittymä. Tällaiset sovellukset soveltuvat alemman tason rutiinien testaukseen, koska graafisten käyttöliittymien teko EPOCilla on sangen hankalaa. ”Todellisiin” sovelluksiin tekstikäyttöliittymää ei voi suositella. Ohjelman toiminta ja ulkonäkö (emulaattorissa) näkyvät kuvassa 3.

Varsinainen koodi on listattu esimerkissä 3. Ohjelma koostuu kahdesta funktiosta: `E32Main` on EPOCin vastine pääohjelmalle, ja `ReadLine` lukee käyttäjän antaman syötteen (rivin) ja palauttaa sitä vastaavan deskriptorin (tyyppiä `TBuf`). Kuten koodin kommenteissakin mainitaan, kyseessä on lähes yltiöyksinkertainen esimerkki, jossa on oiottu joitakin olennaisia mutkia, etenkin virhetarkastelun osalta. Toisin sanoen koodissa ei varauduta sen kummemmin käyttäjän mahdollisesti antamiin ylipitkiin (yli `KLineSize` merkkiä) riveihin kuin dynaamisesti varattavan muistin loppumiseenkaan, vaan kummassakin tapauksessa seurauksena olisi ohjelman kaatuminen. Oikeaan poikkeustilanteiden käsittelyyn ja muistinhallintaan perehdytään seuraavassa luvussa. Koodissa on muutama erillisen selityksen ansaitseva kohta:

Kuva 3. Deskriptoriesimerkin toiminta emulaattorissa.

Esimerkki 3. Deskriptoriesimerkki.

```
// descex.cpp
// Yksinkertaisen väkinäinen deskriptoriesimerkki

#include <e32cons.h>

const TInt KLineSize = 80;

// ReadLine  con  prompt
// Lukee rivillisen merkkejä konsolilta con
// prompt on kehote (esim. "Nimi: " )
TBuf<KLineSize> ReadLine( CConsoleBase* con, const TDesC& prompt ) {
  TBuf<KLineSize> buf;
  TChar key;
  con->Printf( prompt );
  // Luetaan merkki kerrallaan rivinvaihtoon asti
  // Vaarallinen oletus: ei ylipitkiä rivejä
  while( ( key = con->Getch() ) != EKeyEnter ) {
    // Kaiutetaan merkki näytölle
    con->Printf( _L( "%c" ), key );
    buf.Append( key );
  }
  con->Printf( _L( "\n" ) );
  return buf;
}

// Pääohjelma
// Ohjelman toiminta: Käyttäjän kirjoittamat rivit liimataan yhteen
GLDEF_C TInt E32Main() {
  // Literaalivakioita
  _LIT( KPrompt, "\nMerkkijono: " );
  _LIT( KReady, "Valmis." );
  // Luodaan tekstitilan konsoli
  CConsoleBase* con=Console::NewL( _L( "Deskriptoriesimerkki" ),
    TSize( KDefaultConsWidth, KDefaultConsHeight ) );
```

```
    // Aluksi summapuskuri on nollan mittainen
36  // Vaarallinen oletus: muisti ei lopu
    HBufC* cat = HBufC::NewL(0);
38  // Luetaan ja liimataan rivejä aina tyhjään riviin asti
    for( TPtrC line( ReadLine( con, KPrompt) );
40       line.Length();
         line.Set( ReadLine( con, KPrompt ) ) ) {
42    // Varataan lisää tilaa uuden rivin verran
      // ReAllocL luo kokonaan uuden HBufC-instanssin,
44    // joka sisältää vanhan datan.
      // Vaarallinen oletus: muisti ei lopu
46    cat = cat->ReAllocL( cat->Length() + line.Length() );
      // TPtr-instanssin kautta päästään manipuloimaan HBufC:n dataa
48    TPtr ptr = cat->Des();
      ptr.Append( line );
50    *cat = ptr;
      // EPOC-bugi (?): HBufC huomaa datan pituuden muutoksen vasta
52    // ylläolevan sijoituksen jälkeen. Seuraava tosin toimisi:
      // cat->Des().Append( line );
54    // Tulostetaan summarivi
      con->Printf( *cat );
56  }
    // Lopputoimet: dynaamisesti varatun HBufC:n vapautus ja loppukuittaus
58  delete cat;
    con->Printf( KReady );
60  con->Getch();
    return 0;
62  }
```

Rivit 30–31: Makro `_LIT( k, txt )` esittelee literaalivakion `k` ja alustaa sen literaalilla `txt`. Literaalivakio on tyypiltään luokan `TLitC` instanssi, mutta käytännössä tällä tiedolla ei ole merkitystä, koska se voidaan välittää `TDesC`-viitteenä.

Rivi 46: Äkkiseltään voi tuntua oudolta, että `HBufC`:n jäsenfunktio `ReAlloc` palauttaa osoittimen uuteen `HBufC`-instanssiin. Intuition kannalta luonnollisempaa olisi, että tilanvaraukseen liittyvät yksityiskohdat olisi paremmin kätketty eli nimenomaan *saman* olion varaaman tilan kokoa voitaisiin kasvattaa. Tämä ei kuitenkaan käytännössä onnistu, sillä `HBufC` on toteutettu siten, että yhtä instanssia vastaa yksi yhtenäinen muistialue. Suurin syy, miksi näin on tehty, on luultavasti se, että tällöin olion varaamat resurssit voidaan vapauttaa ilman purkajan kutsua eli suoraan `delete`llä. Ainoa mahdollisuus `ReAllocin` toteuttamiselle on varata kokonaan uusi muistialue, kopioida vanhan alueen sisältö ja vapauttaa vanha alue.

Rivit 48–50: Kun tarvittava tila on varattu rivillä 46, niin luettu uusi rivi voidaan liittää osaksi `HBufC`-instanssin `cat` sisältämää merkkijonoa. `HBufC`:n jäsenfunktio `Des` tarjoaa dataan `TPtr`-tyyppisen näkymän. SDK:n dokumentaation (ja mukana tulevien esimerkkien) mukaan `TPtr`-instanssin `ptr` kautta tehtyjen muutosten pitäisi näkyä suoraan myös `cat`issa. Näin ei kuitenkaan

käytännössä tapahdu, vaan rivi 50 on välttämätön. Muutoin rivillä 55 tulostuisi vanha `catin` sisältö ilman uutta riviä. Syynä tuntuisi olevan se, että vaikka merkkijonojen katenointi onnistuukin (osoittaahan `ptr` kuitenkin `catin` data-alueeseen), niin `cat` ei huomaa merkkijonon pituuden muuttumista. Rivi 50 yksinkertaisesti sijoittaa `catin` sisällön siihen itseensä: omituista, turhantuntuista ja tehotonta. Ongelman voisi kiertää sillä, että `ptr`:n luomisen sijaan käytettäisiin aina muotoa `cat->Des()`. Tällöin mainitun kaltaisia ”synkronointiongelmia” ei esiinny.

3.3 Virhetilanteet ja muistinhallinta

EPOC-sovelluksissa tulisi painottaa luotettavuutta ja robustisuutta vieläkin enemmän kuin tavallisissa työpöytäsovelluksissa. Syynä on EPOC-laitteen tyypillisesti suhteellisen pienet resurssit (lähinnä muistin määrä) ja se, että EPOC-filosofian mukaan sovelluksia ei välttämättä ikinä suljeta, vaan ne ovat ajossa jatkuvasti. Käytännössä tämä tarkoittaa sitä, että pienikin muistivuoto voi aikojen kuluessa kumuloitua todelliseksi kiviriipaksi laitteen suorituskyvylle. Sovelluksen tulisikin virhetilanteessa toipua virheestä, jotta sovelluksen käyttö voisi jatkua. Tärkein toipumistoimenpide on muistivuotojen estäminen eli ennen virhetilannetta varattujen ja turhiksi muuttuneiden muistialueiden vapauttaminen.

Virhetilanteiden käsittelyä varten C++ sisältää ns. poikkeusmekanismin. Virhetilanteessa luodaan poikkeus, heitetään (*throw*) se kutsupuussa ylöspäin ja otetaan kiinni (*catch*). Kuinka ollakaan, EPOC ei tue tätä C++:n piirrettä. Sen sijaan se määrittelee oman vastaavantyyppisen poikkeusmekanisminsa. Symbianin perustee asiantilaa sillä, että EPOCin historian alkuhämärissä kääntäjät eivät vielä tukeneet (tulevan) standardin mukaisia poikkeuksia. Lisäksi heidän oman mekanisminsa pitäisi olla varsinaisia poikkeuksia kevyempi ratkaisu.

EPOCin mekanismiin liittyvät tärkeät käsitteet ovat *trap harness* ja *leave*. Noille voisi keksiä kömpelöt suomennoksetkin mutta alkukielisiä termejä käyttämällä vältetään ehkä sekaannukset C++:n varsinaisen poikkeusmekanismin kanssa. Periaate on seuraava: virhetilanteessa heitetään poikkeus eli kutsutaan funktiota `User::Leave`. Tällainen EPOC-poikkeus vastaa C++:n poikkeusta siinä mielessä, että se nousee kutsupuussa ylemmäs. Tarkemmin sanottuna `Leave`lle argumenttina annettu kokonaisluku (ei siis olio kuten yleensä C++:n poikkeuksissa) välittyy ylöspäin. Käsittely tapahtuu vastaavan trap harnessin sisällä. Mikäli poikkeusta ei lainkaan käsitellä, ohjelman suoritus keskeytyy ns. paniikkitilanteeseen. Ohjelmallisesti sovelluksen voi terminoida funktiolla `User::Panic` mutta, kuten aiemmin mainittiin, niin tällaista lähestymistapaa virheisiin tulisi EPOC-sovelluksissa välttää viimeiseen asti.

Esimerkissä 4 näkyy poikkeusmekanismin toiminta kooditasolla ja vieläpä mahdollisimman riisuttuna versiona. Funktio `BehaveL` saa argumenttinaan `TMode`-tyyppisen operaatiotunnisteen ja toimii sen edellyttämällä tavalla. Mikäli annettu tunniste on laiton, heitetään `switch`-lauseen `default`-haarassa poikkeus. EPOCin

Esimerkki 4. EPOCin poikkeusmekanismi.

```
enum TMode { EEat, ESleep, EWait };

void BehaveL( TMode mode ) {

  switch( mode ) {
  case EEat:
    ...
    break;
  case ESleep:
    ...
    break;
  case EWait:
    ...
    break;
  default:
    // Laiton tila: poikkeus
    User::Leave( KErrNotSupported );
  }
  ...
}

GLDEF_C TInt E32Main() {
  ...
  for( TInt i = 0;; i++ ) {
    // Trap harness. erroriin tallettuu virhekoodi
    TRAPD( error, BehaveL( STATIC_CAST( TMode, i ) ) );
    if( error != KErrNone ) {
      // Poikkeus otettu kiinni. Yritetään toipua
      // ja poistutaan silmukasta
      ...
      break;
    }
  }
  ...
  return 0;
}
```

nimeämiskäytäntö edellyttää, että funktion nimen pitää päättyä L:ään, jos funktion suoritus voi päättyä Leave-kutsuun. Tässä tapauksessa Leave-kutsun argumentiksi on valittu valmiiksi määritelty vakio KErrNotSupported. Kyseinen vakio, samoin kuin onnistunutta suoritusta symboloiva KErrNone, on määritelty otsikkotiedostossa e32std.h.

Trap harness on käytännössä TRAPD-makron kutsu, joka saa argumenteikseen muuttujan nimen ja suoritettavan lausekkeen. Makro luo muuttujan ja alustaa sen tarkkailtavan lausekkeen poikkeusarvolla, joka on joko KErrNone (ei poikkeusta) tai Leave-kutsun arvo. Esimerkissä 4 luotu muuttuja on error ja tarkkailtava lauseke BehaveL-funktiokutsu. Pienenä yksityiskohtana voidaan mainita myös se, että EPOCissa C++:n tyyppimuunnosoperaattorit on korvattu makroilla ristiinkääntämisen helpottamiseksi eli STATIC_CAST-makro vastaa toiminnallisuudeltaan static_cast-operaattoria.

Vastuu poikkeusmuuttujan (error) arvon tarkastamisesta sekä mahdollisesta heittämisestä edelleen ylöspäin on tyystin ohjelmoijan harteilla. Toisin sanoen automatiikka ei ole niin kehittynyttä kuin C++:n poikkeusmekanismissa, jossa catch-lauseissa voidaan käsitellä vain osa poikkeuksista, jolloin muut propagoituvat automaattisesti. EPOCissa trap harness ottaa kaikki poikkeukset kiinni. Automaattista propagoitumista tapahtuu ainoastaan silloin, kun poikkeuksen aiheuttava operaatio tehdään trap harnessin ulkopuolella. Näin syntyneet poikkeukset nousevat ylöspäin aina trap harnessiin asti. Esimerkiksi, jos BehaveL kutsuisi (ilman TRAPD:tä) vaikka funktiota DoStuffL, joka aiheuttaisi poikkeuksen, kyseinen poikkeus otettaisiin kiinni vasta pääohjelmassa, jossa itse BehaveL:ää ollaan kutsuttu TRAPD:n sisällä.

Trap harnessista on myös toinen versio: makro TRAP toimii muuten samalla tavalla, mutta poikkeusmuuttuja (error) on täytynyt esitellä jo ennen makrokutsua.

Dynaaminen muistinhallinta yhdessä poikkeusten kanssa voi aiheuttaa sangen kimurantteja tilanteita. Ensinnäkin järjestelmän tulisi huomata muistin loppuminen (eli dynaamisen muistinvarauksen epäonnistuminen) ja virhetilanteessa osata vapauttaa tarpeettomaksi käyneet dynaamisesti varatut alueet. Ensimmäinen vaatimus on toteutettu EPOCissa siten, että new-operaattorista on olemassa globaalisti kuormitettu versio:

```
MyClass* my = new (ELeave) MyClass;
```

Ero ylläolevan ja tavallisen new:n välillä on se, että perus-new palauttaa nollaosoittimen, jos muistivaraus epäonnistuu, kun taas kuormitettu versio heittää poikkeuksen (eli kutsuu Leave-funktiota).

Virheenkäsittelyn yhteydessä tapahtuvan muistinvapauttamisen perusongelma on, että poikkeuksen kiinniottava koodi voi olla kutsupuussa hyvinkin paljon korkeammalla kuin poikkeuksen heittäjä. Tällöin kiinniottaja ei – yleisessä ta-

pauksessa – myöskään pääse käsiksi niihin osoittimiin, jotka osoittavat dynaamisesti varattuihin muistialueisiin. EPOC ratkaisee ongelman siten, että ohjelmoijalle tarjotaan globaali tietorakenne *cleanup stack* (siivouspino), johon kyseisten osoittimien arvot voidaan säilöä. Poikkeustilanteessa pinossa olevia osoittimia vastaavat muistialueet vapautetaan ja `CBase`-tyyppisten osoittimien tapauksessa kutsutaan myös olioiden purkajia. Törmätään taas `CBase`-luokan ja dynaamisen muistinvarauksen symbioosiin: ainoastaan `CBase`-luokasta periytetyillä luokilla saa olla dynaamisesti varattuja jäsenmuuttujia. Muuntyyppiset dynaamisesti varatut muuttujat (kuten esimerkiksi `HBufC`-instanssit) eivät saa edellyttää mitään lopputoimenpiteitä elinkaarensa päättyessä, vaan varatun muistin vapauttaminen sinällään riittää.

Esimerkki 5. `CData`a vastaava otsikkotiedosto.

```
// data.h
// Muistinhallintaesimerkki

#ifndef DATA_H
#define DATA_H

#include <e32base.h>

// class CData
// Periytetty CBase:sta:
//   1. Luodaan aina dynaamisesti
//   2. Virtuaalinen purkaja
class CData : public CBase {
public:
  // Virhekoodi
  static const TInt KErrInvalidId;
  static CData* NewL( TInt id );
  static CData* NewLC( TInt id );
  // Rakentamisen ensimmäinen vaihe
  CData( TInt id );
  // Toinen vaihe
  void ConstructL( const TDesC& prefix );
  // Purkaja
  ~CData();
  void Debug();
  // iId:n pitää olla välillä [-100, 100]
  void CheckIdL();
private:
  TInt iId;
  HBufC* iName;
};

#endif
```

Puidaan virhetilanteisiin ja etenkin siivouspinoon liittyviä seikkoja mahdollisimman yksinkertaisen (mutta tarpeeksi kattavan) esimerkin avulla. Luokka `CData` (esittely esimerkissä 5 ja toteutus esimerkissä 6) on periytetty luokasta `CBase` ja sillä on yksi dynaamisesti varattava jäsen `iName`. Lisäksi kullekin luokan ins-

Esimerkki 6. CData:n toteutus.

```
// data.cpp
// CData:n toteutus

#include "cleanupexample.h"
#include "data.h"

const TInt CData::KErrInvalidId = -1;

CData::CData( TInt id ) : iId( id ) {}

void CData::ConstructL( const TDesC& prefix ) {
  CheckIdL();
  iName = HBufC::NewL( prefix.Length() + 4 );
  iName->Des().Format( _L( "%S %d" ), &prefix, iId );
}

CData::~CData() {
  GetConsole()->Printf( _L( "%d purettu.\n" ), iId );
  delete iName;
}

void CData::CheckIdL() {
  if( iId < -100 || iId > 100 ) {
    User::Leave( KErrInvalidId );
  }
}

void CData::Debug() {
  GetConsole()->Printf( *iName );
  GetConsole()->Printf( _L( "\n" ) );
}

// Luokkafunktiot

CData* CData::NewL( TInt id ) {
  return new (ELeave) CData( id );
}

CData* CData::NewLC( TInt id ) {
  CData* data = new (ELeave) CData( id );
  CleanupStack::PushL( data );
  return data;
}
```

tanssille annetaan tunnistenumero (jäsenmuuttuja `iId`), jonka tulee sopia välille $[-100, 100]$. Huomataankin, että luokan instantioinnissa voi mennä pari asiaa pieleen: ensinnäkin annettu tunnistenumero on laiton, jollin `CheckIdL` heittää poikkeuksen tai `iName`-jäsenmuuttujalle ei saada varattua tilaa (`HBufC::NewL` heittää poikkeuksen). Asiaa tulee vielä astetta monimutkaisemmaksi, kun havahdutaan EPOCin ehdottomaan sääntöön: rakentaja *ei saa koskaan* heittää poikkeusta. Problematiikan ratkaisun avain on kaksivaiheinen rakentaminen. Ensimmäisessä vaiheessa kutsutaan luokan rakentajaa ja tehdään ne alkutoimet, jotka eivät voi heittää poikkeusta. Toinen vaihe tehdään erillisen jäsenfunktion avulla, joka tekee loput alustukset. EPOC-käytäntönä on antaa tälle toisen vaiheen funktiolle nimeksi `ConstructL`. Kaksivaiheisuudesta on se etu, että rakentajakutsun jälkeen olio on jo instantioitu, joten siihen voidaan osoittaa ja vielä tärkeämpää: osoitin voidaan työntää siivouspinoon. Jos `ConstructL` tämän jälkeen ”kaatuu” poikkeukseen, niin siivouspino osaa kutsua olion purkajaa, jolloin osittainkin rakennetun (eli `ConstructL` ei ole ehtinyt alustaa kaikkia jäsenmuuttujia ennen poikkeusta) olion purkajaa kutsutaan ennen tilan vapauttamista. `CBase` on toteutettu niin, että purkajan kutsu alustamattomienkin dynaamisten jäsenmuuttujien (eli osoittimien) tapauksessa on turvallinen. Käytännössä tämä tarkoittaa sitä, että `CBase`n rakentaja täyttää koko olion varaaman tilan nollilla, jolloin kaikkien jäsenmuuttujien alkuarvokin on nolla ja nollaosoittimen ”tuhoaminen” `delete`llä on sallittua.

Luokkafunktiot `NewL` ja `NewLC` ovat EPOC-luokille usein toteutettuja mukavuusfunktioita. Niiden välinen ero on siinä, että C-päätteinen (*cleanup*) tallettaa luotua oliota vastaavan osoittimen siivouspinoon. Luokan toteutus on muutoin aika suoraviivainen (joskin väkinäinen). Merkille pantava piirre on nimenomaan se, että siivouspinon olemassaolo ei `NewLC`:tä lukuun ottamatta näy. Syy tähän on se, että kaikki dynaamisesti varatut oliot ovat jäsenmuuttujia, joten niihin päästään aina käsiksi. Mikäli jäsenfunktiot loisivat väliaikaisia dynaamisia muuttujia, tilanne olisi toinen…

Esimerkissä 7 päästään varsinaiseen asiaan: `CData`-olioiden käyttämiseen. Otsikkotiedosto `cleanupexample.h` sisältää ainoastaan `GetConsole`-funktion esittelyn, joten sitä ei tarvinne listata erikseen. Ohjelman toiminta näkyy kuvassa 4.

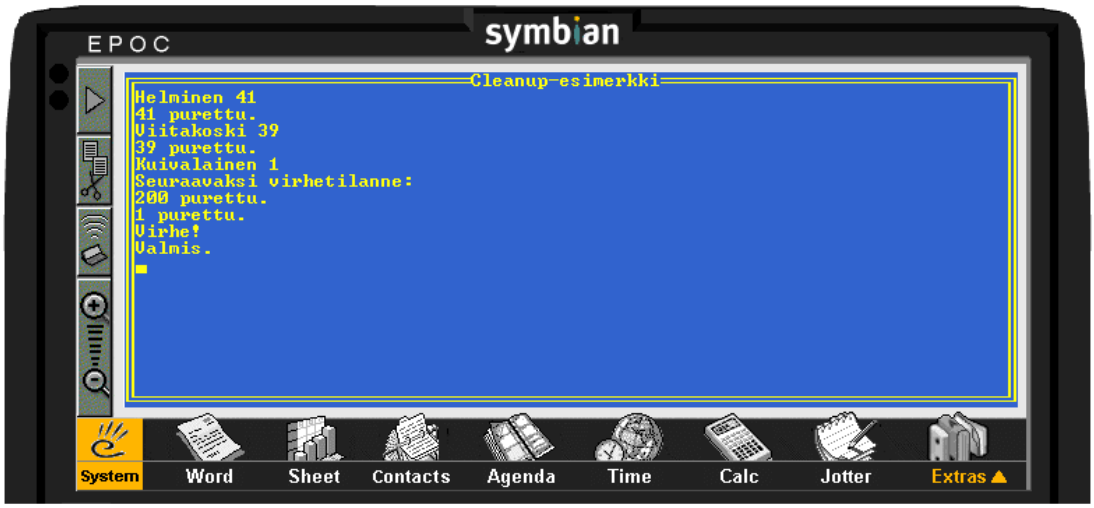
Esimerkki 7. Esimerkki siivouspinon käytöstä.

```
#include "data.h"
#include "cleanupexample.h"

// GetConsole
CConsoleBase* GetConsole() {
  static CConsoleBase* console = Console::NewL( _L( "Cleanup-esimerkki" ),
                                                TSize( KDefaultConsWidth,
                                                       KDefaultConsHeight ) );
  return console;
}
```

```
// FooL
// Esimerkkifunktio. Normaatilanteessa päättyy poikkeukseen ja
// siivouspinoon jää oliot third ja wrong
void FooL() {
  CData* first = new (ELeave) CData( 41 );
  // ConstrucL:stä voi singahtaa poikkeus: first pinoon
  CleanupStack::PushL( first );
  first->ConstructL( _L( "Helminen" ) );
  first->Debug();
  // first:iä ei enää tarvita. Popataan ja tuhotaan
  CleanupStack::Pop();
  delete first;
  // NewL-idioomi
  CData* second = CData::NewL( 39 );
  CleanupStack::PushL( second );
  second->ConstructL( _L( "Viitakoski" ) );
  second->Debug();
  // Pop ja delete yhdessä
  CleanupStack::PopAndDestroy();
  // Luonti ja PushL yhdistetty
  CData* third = CData::NewLC( 1 );
  third->ConstructL( _L( "Kuivalainen" ) );
  third->Debug();
  // Rakennetaan laiton instanssi (id liian suuri)
  // ja pistetään se pinoon.
  CData* wrong = CData::NewLC( 200 );
  // ConstructL tulee heittämään poikkeuksen
  // Kerrotaan ensin mitä ollaan tekemässä
  GetConsole()->Printf( _L( "Seuraavaksi virhetilanne:\n" ) );
  wrong->ConstructL( _L( "Halkidis" ) );
  // Loput rivit ovat turhia (niitä ei ikinä suoriteta) mutta pistetään
  // ne mukaan symmetrian takia. Tuotantokoodissakin poikkeustilanteet
  // ovat useimmiten tahattomia.
  wrong->Debug();
  CleanupStack::PopAndDestroy();   // wrong
  CleanupStack::PopAndDestroy();   // third
}

// Pääohjelma
// Kutsutaan funktiota FooL ja otetaan syntynyt poikkeus kiinni
GLDEF_C TInt E32Main() {
  // Luodaan uusi siivouspino
  CTrapCleanup* cleanupstack = CTrapCleanup::New();
  TRAPD( error, FooL() );
  if( error != KErrNone ) {
    GetConsole()->Printf( _L( "Virhe!\n" ) );
  }
  GetConsole()->Printf( _L( "Valmis.\n" ) );
  GetConsole()->Getch();
  // Siivotaan siivouspino
  delete cleanupstack;
  return 0;
}
```



Kuva 4. Siivouspinoesimerkin toiminta.

Osa esimerkin koodista ansaitsee lähemmän tarkastelun:

Rivi 54: Jotta siivouspinoa voitaisiin käyttää, täytyy se ensin luoda. Siivouspinon käsittely tapahtuu kuitenkin aina luokan `CleanupStack` luokkafunktioiden avulla (esim. `PushL`). Itse asiassa siivouspinotkin muodostavat pinon: viimeksi luotu (eli pinon päällimmäinen) on käytössä. Kun pino tuhotaan, niin uudeksi siivouspinoksi tulee pinojen pinossa seuraavana oleva.

Rivit 16–23: `first` on paikallinen muuttuja, joten se tulee tallettaa siivouspinoon (rivi 18) ennen mahdollista poikkeusta (rivi 19). Koska `Debug`-jäsenfunktio ei voi heittää poikkeusta, `first` voitaisiin popata pinosta jo heti rivin 19 jälkeen. Tässä tapauksessa on kuitenkin haluttu yhdistää `Pop`-kutsu olion tuhoamiseen (rivit 22 ja 23).

Rivi 25: `NewL`-idioomi tekee täsmälleen saman asian kuin rivin 16 `new`, tosin luonnollisesti eri argumentilla.

Rivi 30: `PopAndDestroy` yhdistää `Popin` ja `deleten`.

Rivi 37–41: Luotava ja pinoon laitettava olio on laiton, koska sen tunnistenumero (rakentajan argumentti) on liian suuri. `ConstructL`-kutsu aiheuttaa poikkeuksen. Tällöin siivouspinossa on kahden olion osoittimet: `thirdin` ja `wrongin`.

Rivit 45–47: Näitä rivejä *ei ikinä* suoriteta.

Vaikka cleanup stack -mekanismi (alkujärkytyksen jälkeen) vaikuttaakin suhteellisen käytettävältä ja täysjärkiseltä systeemiltä, voi käytännössä silti törmätä omituisiin ongelmiin. Ainakin emulaattori tuntuu kaatuvan kovinkin herkästi ja mitä oudoimmissa tilanteissa (`CTrapCleanup::New` ja `CleanupStack::PushL` samassa

funktiossa, PushL ilman Popia samassa funktiossa, PushL:n sisältävän funktion kutsu ilman TRAPD:tä jne.). Mitään pomminvarmoja kaatumisenesto-ohjeita on kuitenkin hyvin vaikea antaa, koska mitään järkeenkäypää yhteistä tekijää eri kaatumisille ei – ainakaan toistaiseksi – ole löytynyt.

CleanupStack::PushL-funktioon liittyy pari mielenkiintoista seikkaa. Ensinnäkin siivouspinon toteutus takaa, että vaikka PushL aiheuttaisi poikkeuksen, niin sen argumenttina annettu osoitin on laitettu pinoon ennen poikkeuksen heittämistä. Tämähän tarkoittaa sitä, että siivoustoimenpiteet yksinkertaistuvat, koska epäonnistunutta PushL:ää ei tarvitse käsitellä erikoistapauksena. Käytännön toteutus on sellainen, että PushL osoittimen tallentamisen lisäksi myös varaa tilaa seuraavalle osoittimelle eli itseasiassa PushL:n heittämä poikkeus tarkoittaakin sitä, että pinossa ei ole enää tilaa *seuraavaksi* talletettavalle osoittimelle. Toinen mielenkiintoinen asia liittyy PushL:n kuormittamiseen. Tarkemmin sanottuna PushL:stä on kolme eri versiota, joiden saamat argumentit ovat tyyppiä const CBase*, const TAny* ja TCleanupItem. TCleanupItem-instanssi kapseloi sekä pinoon talletettavan osoittimen, että siihen liittyvän puhdistusoperaation. Näin geneerinen mekanismi voi olla käyttökelpoinen esimerkiksi silloin, kun käsitellään oliota, jota ei olla periytetty CBasesta mutta joka tarvitsee silti joitain erityisiä siivoustoimenpiteitä pelkän muistinvapauttamisen lisäksi. CleanupStack::Popiakin on kuormitettu sen verran, että sille voidaan antaa argumentiksi popattavien osoittimien lukumäärä.

Muistinhallinnan debuggausta varten SDK sisältää erilaisia mekanismeja, joilla voidaan simuloida esimerkiksi muistin loppumista.

4 Graafiset käyttöliittymät

Graafisten käyttöliittymien rakentamiseen tarkoitetut kirjastot sisältävät tavallisesti seuraavankaltaisia osia:

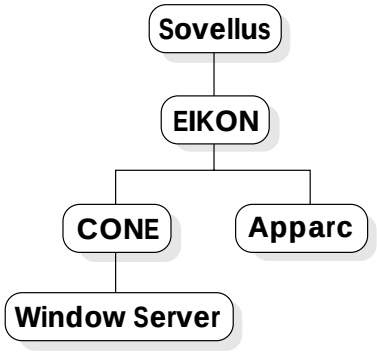
Käyttöliittymäkomponentit. Tunnistettavat, usein pitkälle erikoistuneet, käyttöliittymäkomponentit, joita joissain yhteyksissä kutsutaan myös widgeteiksi. Tuttuja esimerkkejä ovat painonapit, tekstikentät, erilaiset valikot, jne. Ohjelmoijalle tarjottu komponenttivalikoima määrää tehtävien käyttöliittymäratkaisujen luonteen hyvinkin pitkälle, koska omien uusien komponenttien rakentaminen on usein kovin työlästä ja saavutettu lopputulos voi sotia totuttua käytettävyyttä vastaan. Toisaalta valmiiden komponenttien erikoistamisen helppous ja monipuolisuus voi olla myös yksi käyttöliittymäkirjaston valinnan kriteeri.

Alustat. Edellä mainitut komponentit ladotaan osaksi jotakin erityistä alustakomponenttia (*container*). Alustan velvollisuuksiin kuuluu säilyttää komponenttien keskinäinen järjestys (*layout*) johdonmukaisena silloinkin, kun alustasta poistetaan komponentteja tai alustan koko muuttuu dynaamisesti. Laajasti ajatellen alustoihin voidaan sisällyttää myös erilliset ikkunakomponentit.

Tapahtumankäsittely. Käyttöliittymien toiminnallisuus nojautuu paljolti tapahtumiin (*events*). Esimerkiksi nappikomponentin painaminen aiheuttaa tapahtuman, joka välittyy ohjelmalle. Tämän välitysmekanismin toteutus samoin kuin tapahtuman mallintaminenkin ovat kirjastokohtaisia. Yksinkertaisimmillaan järjestelmässä on yksi massiivinen tapahtumankäsittelyfunktio, joka saa argumentikseen tapahtumatyypin identifioivan lukuarvon ja mahdollisesti joitakin tarkentimia. Tällöin kyseisen funktion toteutus luisuu helposti kohti infernaalisen pitkää `switch`-lausetta, jonka yhteydessä joustavuudesta ja ylläpidettävyydestä ei voida vakavasti puhua.

Sofistikoituneemmat ratkaisut pyrkivät jakamaan tapahtumankäsittelyn eri osapuoliin, jolloin ohjelman eri kohdissa voidaan käsitellä sinne loogisesti kuuluvat tapahtumat. Perinteisiä esimerkkejä tästä lähestymistavasta ovat erilaiset callback-mekanismit. Oliopohjaisissa kirjastoissa tapahtumat kannattaa luonnollisesti esittää olioina ja vieläpä siten, että eri tapahtumatyypit on jaettu eri luokkiin (jotka tosin voivat kuulua samaan luokkahierarkiaan).

Valmiit rutiinit. Eri käyttöliittymäkirjastoja voidaan jaotella myös niiden helppokäyttöisyyden ja intuitiivisuuden mukaan. Etenkin tutustuttaessa uuteen kirjastoon on tärkeää, että ohjelmoija saa suhteellisen helposti jotain näkyvää aikaiseksi. Pitemmällä tähtäimellä on toivottavaa, että ohjelmoija voi keskittyä ohjelman toiminnallisuuden kannalta olennaisiin käyttöliittymän piirteisiin ilman tappelua käyttöliittymän alkeellisimman perustoiminnallisuuden parissa. Yleensä kirjastot pitävät huolen ainakin siitä, että käyttöliittymäkomponentit piirretään tarvittaessa uudestaan, mutta ongelmia voi



Kuva 5. EIKON EPOC-hierarkiassa [7].

tulla kuitenkin jo siinä vaiheessa, kun on selvitettävä, miten johonkin alustaan kiinnitetyt komponentit saadaan ensimmäisen kerran näkyviin. Tyypillinen piirtoalustakomponentti (*canvas*) on taas sellainen, jonka päivittäminen on tyystin ohjelmoijan harteilla. Mainitsemisen arvoinen poikkeus on Tk-kirjasto[1], joka sisältää tuen mm. erilaisten piirtoelementtien (viivojen, ympyröiden, ...) automaattiselle päivittämiselle ja erilaisille operaatioille (siirtämiselle, kytkemiselle tapahtumiin, leiketaululle (*clipboard*), ...).

EPOCin käyttöliittymäkirjasto on nimeltään EIKON. Se tarjoaa käyttöliittymäkomponentit ja niiden käsittelyyn tarvittavat rutiinit. Lisäksi EIKONin kautta päästään käsiksi CONE- ja Apparc-sovelluskehyksen rajapintoihin, koska EIKON on näiden kahden erikoistus (ks. kuva 5). Tämänkaltainen hierarkia aiheuttaa jonkin verran ongelmia, koska ei ole mitenkään ilmeistä minkä osan reviiriä jokin haluttu toiminto on. Ongelmaa kärjistää entisestään jopa EPOCin mittakaavassa riittämätön dokumentaatio: läheskään kaikkia hyödyllisiä luokkia ei dokumentaatiosta löydy, jolloin ohjelmoija joutuu perehtymään SDK:n mukana tuleviin lähes kommentoimattomiin otsikko- ja lähdekooditiedostoihin. Samoin kunnolliset esimerkit ovat kiven alla.

4.1 CONE

CONEn roolina on tarjota korkeamman tason rajapinta Window Serverin sisältämille palveluille (ks. kuva 5). Koska EIKON niputtaa yhteen sekä omat uudet luokkansa että CONE-luokat, voi ohjelmoijalla olla pieniä vaikeuksia eri luokkien (ja laajempienkin rajapintakokonaisuuksien) vastuualueiden hahmottamisessa. Tärkeimmät CONE-luokat:

`CCoeControl`. EPOC-terminologiassa kontrolli (*control*) tarkoittaa käyttöliittymien yhteydessä suorakaiteenmuotoista näytön aluetta, joka voi ottaa vastaan tapahtumia. `CCoeControl` on kontrollien kantaluokka, josta myös käyttöliittymäkirjastokohtaiset kontrollit (widgetit) on periytetty. EIKONin ta-

[1] `http://www.scriptics.com`

pauksessa näitä kontrolleja ovat mm. CEikLabel, CEikCommandButton ja CEikEdwin.

Ohjelmoija voi myös periyttää CCoeControlista omia kontrollejaan. CCoeControl tarjoaa laajan rajapinnan kontrollin alustamiseen, piirtämiseen ja tapahtumankäsittelyyn. Kontrolli voi toimia myös alustana (*compound control*), jolloin se sisältää muita kontrolleja ja pitää huolen esimerkiksi asemoinnista ja tapahtumien välittämisestä. Kontrollit jaotellaan kahteen luokkaan myös sen mukaan, miten ne sijoittuvat ikkunahierarkiassa. Ikkunalliset (*window-owning*) kontrollit ovat kontrolleja, jotka ovat – Window Serverin kannalta – samassa asemassa kuin niiden ikkunakin, eli toisin sanoen ne omistavat ikkunan (esim. valikot (*menu*), työkalunauha (*toolband* tai *top toolbar*) ja -palkki (*toolbar*), dialogit). Ikkunattomat (*non-window-owning*) kontrollit toimivat ikkunan osana, eli ikkuna voi sisältää useammankin ikkunattoman kontrollin. Tähän kategoriaan kuuluu suurin osa kontrolleista (esim. CCoeControl-kohdassa mainitut EIKON-kontrollit).

CCoeEnv. Erilaisiin käyttöliittymän ulkopuolisiin palveluihin päästään käsiksi CCoeEnv-luokan avulla. Luokka sisältää erilaisia hyödyllisiä funktioita esimerkiksi resurssitiedostojen käsittelyyn. Useimmiten ohjelmoija kuitenkin käyttää luokan CCoeControl lapsiluokille näkyvää iEikonEnv-jäsenmuuttujaa, joka on luokan CEikonEnv-instanssi ja toimii myös CCoeEnv-wrapperina. Hyödyllisyysfunktioiden lisäksi CCoeEnv tarjoaa rajapinnan Window Serverin kanssa kommunikointiin EPOCin oman asiakas-palvelinyhteysmekanismin avulla.

CCoeAppUi. Kullakin CONE-sovelluksella on täsmälleen yksi sovelluskohtainen käyttöliittymäkäytäntö (*app UI*). Käytäntö vaikuttaa lähinnä näppäinten painallusten (*key events*) käsittelyyn. EIKON-sovellusten käyttöliittymäkäytäntö määräytyy luokan CEikAppUi mukaan, jonka pohjana on CONE-luokka CCoeAppUi. Näppäinpainallusten käsittelyn suhteen tämä tarkoittaa sitä, että eri kontrollit pääsevät näppäintapahtumiin käsiksi ns. kontrollipinon kautta (*control stack*). Pinoa voidaan ajatella eräänlaisena suodattimena: näppäinpainallusta vastaavaa tapahtumaa tarjotaan vuorotellen kullekin pinon kontrollille, kunnes jokin niistä käsittelee sen. Pinon alkioille voidaan antaa prioriteetteja, jotka vaikuttavat kontrollien järjestykseen. Kooditasolla näppäintapahtumat käsitellään seuraavasti:

1. Näppäintapahtuman käsittelevä kontrolli lisätään kontrollipinoon funktiolla CCoeAppUi::AddToStackL. Tavallisesti tämä tehdään sovelluksen CEikAppUi-toteutuksen kaksivaiheisen rakentajan ConstructL-jäsenfunktiossa.
2. Kun jotakin näppäintä painetaan sovelluksen ollessa valittuna, kutsutaan kontrollipinon ensimmäisen kontrollin OfferKeyEventL-funktiota.
3. Jos kontrolli käsittelee tapahtuman, se palauttaa paluuarvona tunnisteen EKeyWasConsumed. Mikäli paluuarvo onkin EKeyWasNotConsumed, tapahtumaa tarjotaan kontrollipinon seuraavalle alkiolle, jne.

Vaikkakin kontrollipinon toiminta ei kovin mutkikkaalta kuulostakaan, voidaan silti argumentoida, että mekanismin selkeydessä olisi toivomisen varaa. On sangen hämäävää, että näppäintapahtumat käsitellään tyystin eri tavalla kuin muut. Lisäksi kontrollipinomekanismin tietynlainen paljastaminen luokan rajapinnassa ei välttämättä auta asiaa. Jäsenfunktion `AddToStackL` yhteys näppäinpainallusten kiinnisaamiseen (eli kontrollin `OfferKeyEventL`-jäsenfunktioon) ei ole mitenkään intuitiivinen. Voidaan vain arvella, kuinka monta tuntia on hukattu sellaisten EPOC-sovellusten parissa, joissa on toteutettu `OfferKeyEventL` muttei ole hoksattu ”rekisteröidä” kontrollia `AddToStackL`:llä.

`MCoeControlObserver`. Kuten luvussa 3.2 kerrottiin, periytymällä M-luokasta luokka sitoutuu noudattamaan M-luokan määrittelemää protokollaa eli toteuttamaan sen sisältämät funktiot. `MCoeControlObserver`-protokollalla voidaan tietyntyyppiset tapahtumat välittää keskitetysti yhdelle kontrollille (protokollan tarjoajalle). Yleensä tämä tarkoittaa sitä, että alustakontrolli tarkkailee sisältämiään kontrolleja seuraavalla tavalla:

1. Alusta periytetään luokasta `MCoeControlObserver`.
2. Kullekin tarkkailtavalle kontrollille kutsutaan sen (`CCoeControl`ista perittyä) `SetObserver`-funktiota, jonka argumentiksi annetaan osoitin alustaan.
3. Tietyntyyppiset tapahtumat välittyvät alustan kontrolleista suoraan alustalle eli ne aiheuttavat alustan `HandleControlEventL`-funktion kutsun. Tällaisia tapahtumia ovat mm. kontrollin valintaan (*focus*) ja tilan muuttumiseen (esim. napin painaminen) liittyvät tapahtumat. `HandleControlEventL` saa argumentikseen tapahtuman generoineen kontrollin ja tapahtumatyypin tunnisteen, joka on tyyppiä `TCoeEvent`.
4. Kontrollin jäsenfunktion `ReportEventL` avulla voidaan välittää tapahtuma tarkkailijalle ohjelmallisesti.

`MCoeControlContext`. Grafiikkakonteksti (*graphics context*) voi olla yhteinen useamman kontrollin kanssa, jolloin on helpompi saavuttaa sovelluksen kauttaaltaan yhtenäinen ulkoasu. Kontekstin jakaminen:

1. Kontekstin määräävä luokka (yleensä alusta) periytetään jostain `MCoeControlContext`in lapsiluokasta. Yksi kantaluokkakandidaatti tarjotaan valmiina: `MCoeControlBrushContext`.
2. Kontekstin välittäminen muille kontrolleille tapahtuu `SetContainerWindowL`-jäsenfunktiokutsun yhteydessä, jos kontrolli jakaa alustansa kontekstin. Kontekstikontrolli voidaan kertoa myös eksplisiittisesti sopivalla jäsenfunktiolla (esim. `SetControlContext`) tai asettamalla (`CCoeControl`ista periytetty) jäsenmuuttuja `iContext`.
3. Myöhemmin kontrollit pääsevät käsiksi kontekstiin `iContext`-jäsenmuuttujan kautta.

Ärsyttävintä CONEssa ja EPOCin GUI-mekanismeissa yleensä on yllättävä alkeellisuus. Vaikka suuret, arkkitehtuuritason linjat tuntuvat äkkilukemalta hyvinkin fiksuilta, kooditasolla joudutaan kuitenkin painiskelemaan epäolennaisten yksityiskohtien parissa ainakin silloin, kun ei satuta vahingossa törmäämään dokumentaation- tai koodinpätkään, jossa on toteutettu valmiiksi vaadittavat yksityiskohdat. Graafisten käyttöliittymien tapauksessa moisen onnenpotkun todennäköisyys on EPOCissa aika pieni.

Esimerkkinä kohdattavista vaikeuksista voitaisiin ajatella tilannetta, jossa halutaan saada näytölle yksi tai useampi tekstikenttä. Tekstikenttiä tulisi voida editoida. Tuntuisi luontevalta kiinnittää tekstikentät johonkin alustakontrolliin. EIKON sisältää joitakin kontrolleja, kuten `CEikForm` ja `CEikScrollableForm`, jotka mahdollisesti voisivat toimia alustoina, mutta dokumentaatio sivuuttaa ne enintään maininnalla ja esimerkitkään eivät ole kovin selkeitä, joten ohjelmoija alistunee tekemään alustan itse. Tällöin joudutaan koodaamaan useampi matalan tason rutiini: alustan täytyy ainakin osata kertoa montako kontrollia se sisältää (`CountComponentControls`), ja tarjota mahdollisuus niiden indeksoimiseen (`ComponentControl`).

Kun tekstikentät on saatu kiinnitettyä alustaan ja näkyviin näytölle, huomataan, että kenttien editoiminen ei onnistu. Ohjelmoijan täytyy käsitellä yksittäiseen tekstikenttään liittyvät tapahtumat (valinta ja kirjoittaminen) ja välittää ne tekstikenttäkontrollille. Samalla täytyy muistaa rekisteröidä alusta kontrollipinoon `CEikAppUi`-luokan `AddToStackL`-metodilla. Lisää ”käsityötä” kertyy, jos sovelluksen halutaan noudattavan yleisiä käytettävyyskonventioita (esim. tekstikentästä toiseen siirtyminen tabulaattorilla tai nuolinäppäimillä).

Jossain vaiheessa ohjelmoijaa alkaa kalvaa mieleen hiipivä epäilys siitä, että EPOC sisältää joitain fiksumpia, korkeamman tason mekanismeja käyttöliittymien tekemiseen. Tähän hartaaseen toiveeseen on helppo yhtyä, mutta toistaiseksi dokumentaatio ainakin tuntuu vaikenevan tällaisesta mahdollisuudesta tyystin.

4.2 EIKON-sovellus

EIKON-sovellus koostuu vähintään neljästä oliosta:

Sovellusolio. Luokan `CEikApplication` instanssi, joka voi tarjota sovelluksenlaajuisia palveluita. Olion pääasiallinen tehtävä on dokumentin luonti.

Dokumentti. Luvussa 2.1 käytiin lyhyesti läpi MVC-suunnittelumalli ja mainittiin sen kytkös EPOCiin. EIKON-sovelluksissa dokumentti eli luokan `CEikDocument` instanssi vastaa MVC:n mallia. Toisin sanoen se kapseloi sovelluksen sisäisen tilan. Dokumentin tehtäviin kuuluvat yleensä myös mallin tallettamiseen ja lukemiseen liittyvät toimenpiteet. EPOC vaatii, että jokaisella EIKON-sovelluksella on dokumentti, vaikkei sille olisikaan sovelluk-

sen kannalta mitään järkevää tulkintaa. Tämä sen takia, että dokumentti puolestaan luo käyttöliittymän (*app UI*).

Käyttöliittymä. Tässä käyttöliittymällä tarkoitetaan luokan `CEikAppUi` instanssia, joka on osaltaan vastuussa käyttöliittymän toiminnallisuudesta (ks. edellinen luku). MVC:n kannalta tämä olio on lähinnä kontrolleria.

Näkymät. Kaikkia muita osia on yksi kappale, mutta näkymiä (luokka `CEikAppView`) voi olla useampiakin. Näkymien luonti on käyttöliittymän vastuulla.

Sovellusten muokattavuuden kannalta on yleensä toivottavaa, että sovelluksen logiikka, käyttöliittymä ja muut piirteet olisi mahdollisimman hyvin eristetty toisistaan. Lisäksi varsinaisen koodin ei tulisi ottaa kantaa epäolennaisiin yksityiskohtiin, kuten esimerkiksi käyttöliittymäkontrollien teksteihin. Paras vaihtoehto olisi, jos järjestelmä tukisi sellaisia mekanismeja, joilla sovellusta voitaisiin konfiguroida helposti ilman ohjelman uudelleen kääntämistä. EPOCissa tällainen mekanismi ovat resurssitiedostot, jotka mainittiin lyhyesti jo luvussa 3.1.

Rakennetaan nyt yksinkertainen EIKON-sovellus, jolla voidaan helpohkosti laskea painoindeksi. Painoindeksin kaava on

$$\text{painoindeksi} = \frac{\text{paino}}{\text{pituus}^2} = \frac{\text{kg}}{\text{m}^2}$$

Saatu indeksi tulkitaan siten, että normaalipainoisen henkilön indeksi sijoittuu välille 20–25. Pienemmät tai suuremmat lukemat viittaavat ali- ja ylipainoisuuteen. Sovelluksen käyttöliittymä näkyy kuvassa 6.

Esimerkki koostuu seuraavista (ohjelmoijan kirjoittamista) tiedostoista:

`fats.mmp`. `makmake`-työkalun tarvitsema projektitiedosto. Tiedosto on sama kuin aiemmin sivulla 13 ollut esimerkki 1.

`fats.rss`. Ohjelman ulkoasuun (lähinnä valikon ja työkalupalkin sisältöön sekä näkyviin teksteihin) vaikuttava resurssitiedosto. Esimerkkilistaus 10.

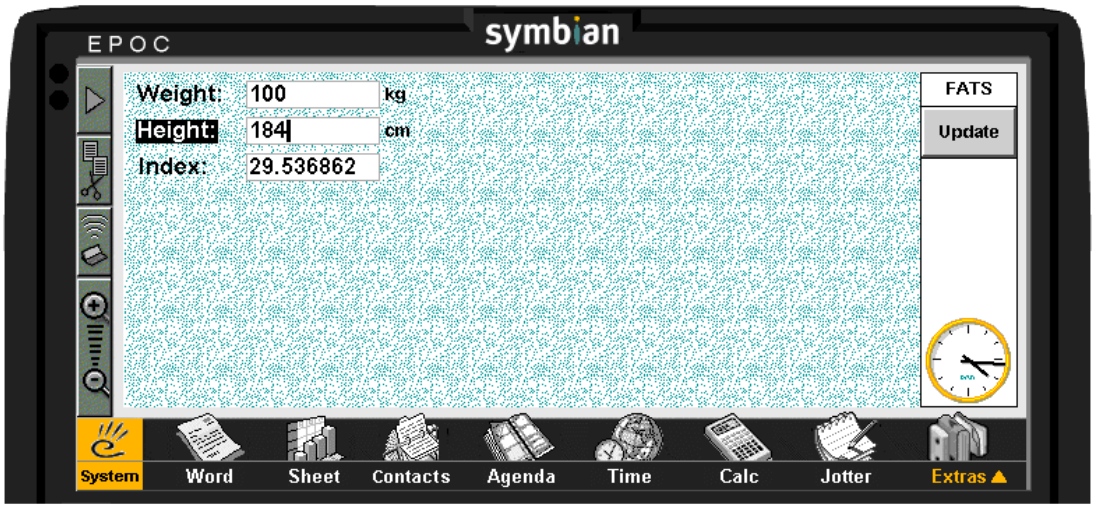
`fats.hrh`. Otsikkotiedosto, jossa määritellään resurssitiedostossa käytettyjä lukuvakioita. Esimerkkilistaus 11.

`fats.h`. Otsikkotiedosto, jossa on kaikkien rakennettujen luokkien esittelyt. Esimerkkilistaus 12.

`fats.cpp`. Sovelluskehyksen vaatimien erikoistusluokkien lähdekoodit näkymää lukuun ottamatta. Esimerkkilistaus 13.

`fatsappview.cpp`. Näkymän lähdekoodi. Esimerkkilistaus 14.

Esimerkit löytyvät (`fats.mmp`:tä lukuun ottamatta) julkaisun liitteestä A alkaen sivulta 57.



Kuva 6. Painoindeksisovellus emulaattorissa.

Resurssitiedosto `fats.rss`:n formaatti on suhteellisen selkeä: määriteltävä asia on oma lohkonsa ja erityyppiset lohkot sisältävät erilaisia parametreja. Syntaksiltaan konfiguroiminen muistuttaa tavallaan C++:n tietueiden käsittelyä. Käydään resurssitiedosto systemaattisesti läpi soveltuvin osin:

Esim. 10, s. 57: `fats.rss`

Rivi 3: `NAME`-lause identifioi resurssitiedoston. Jokaisella EIKON-sovelluksella on vähintään kaksi resurssitiedostoa: sovelluksen ”oma” ja EIKONia konfiguroiva `eikon.rss`. Nimen maksimipituus on neljä merkkiä, ja `NAME` on tiedoston ensimmäinen lause.

Rivi 18: `RSS_SIGNATURE`n pitää löytyä ja sen tulee olla tyhjä. Toisin sanoen EIKON tarvitsee sitä sisäisesti mutta ohjelmoijan ei kannata asiaa sen kummemmin murehtia.

Rivi 20: `TBUF` kertoo sovelluksen dokumenttia vastaavan oletustiedoston nimen. Koska painoindeksi on ”tilaton”, niin resurssi on tyhjä.

Rivit 22–26: `EIK_APP_INFO`n mahdolliset kentät ovat `hotkeys` (näppäinoikotiet), `menubar`, `toolbar` ja `toolband` (ylätyökalupalkki).

Rivit 28–32: Näppäinyhdistelmä Ctrl-e vastaa komentoa `EEikCmdExit`.

Rivit 34–47: Valikoiden määrittely on hyvin suoraviivaista. Tässä tapauksessa valikoita on vain yksi (File) ja siinäkin vain yksi alkio (Exit).

Rivit 49–81: Kuten kuvasta 6 näkyy, sovelluksen työkalupalkki koostuu useammasta kontrollista: ylinnä on `EEikCtFileNameLabel`, jota painamalla aukeaa dialogi, jossa näkyy avoinna olevat tiedostot/sovellukset. Seuraavana on komentoa `ECmdUpdate` vastaava Update-nappi. Tämän jälkeen tyhjää tai tarkemmin venyvä täytekontrolli. Vastaavantyyppinen kontrolli on myös alimpana palkissa. Täytteiden välissä on kello.

Rivit 83–85: TBUF-kontrolleja käytetään yleisestikin tekstivakioiden määrittelyyn. Aiemmin mainittu TBUF on erityisasemassa ainoastaan sijaintinsa takia. Toisin sanoen ei ole samantekevää, missä järjestyksessä eri asiat resurssitiedostossa määritellään. Etenkin resurssien NAME, RSS_SIGNATURE, ensimmäinen TBUF ja EIK_APP_INFO kanssa täytyy olla tarkkana.

Otsikkotiedosto fats.hrh ainoastaan antaa sisällön resurssitiedostossa viitatuille resurssien tunnisteille. Painoindeksin tapauksessa tällaisia ovat ECmdUpdate ja EFatsFileName. Komentotunniste EEikCmdExit on EIKONin valmiiksi määrittelemä. Jotta vältyttäisiin tunnistetörmäyksiltä, tulisi muistaa, että EIKON on varannut itselleen tunnisteet väliltä 0x100–0x1ff.

Ohjelmoijan kannalta painoindeksisovelluksen ”pihvi” eli varsinaisen sovelluslogiikan sisältävä osa on luokka CFatsAppView, jota tapahtumankäsittelyssä jonkin verran auttelee luokka CFatsAppUi. Muiden fats.h:ssa esiteltyjen luokkien tehtävänä on ainoastaan tarjota mekanismit, joilla logiikka saadaan ”ripustettua” osaksi järjestelmää. Olioiden luomisen kannalta sovelluksen käynnistäminen tapahtuu seuraavasti:

EPOC $\xrightarrow{\text{kutsuu}}$ NewApplication $\xrightarrow{\text{luo}}$ CFatsApplication $\xrightarrow{\text{luo}}$ CFatsDocument $\xrightarrow{\text{luo}}$ CFatsAppUi $\xrightarrow{\text{luo}}$ CFatsAppView

Muiden paitsi näkymäluokan CFatsAppView toteutus on tiedostossa fats.cpp. Siinä missä aiemmin puhuttiin pihvistä, tutkaillaan tällä kertaa luurankoja, eli luokkien toiminnallisuus lähestulkoon rajoittuu ”seuraavan” olion luomiseen. On joukossa toki joitakin erityishuomiota ansaitsevia kohtia:

Rivit 5–7: NewApplication lienee EIKON-sovelluksen lähin vastine pääohjelmalle. Kyseessä on nimenomaan vastine, sillä EIKON-sovellus on oikeasti dynaaminen kirjasto eikä siinä mielessä itsenäinen sovellus kuin voisi ajatella. EXPORT_C-tarkennin kertoo, että kyseistä funktiota voidaan kutsua DLL:n ulkopuolelta (muiden DLL:ien tai sovellusten toimesta). Termi ”export” viittaa tässä nimenomaan kutsujaan eli funktio on viety sen ulkopuolelle.

Esim. 13, s. 60: fats.cpp

Rivit 9–11: SDK:n dokumentaatio sivuttaa tämän funktion seuraavankaltaisella maininnalla: funktion pitää olla olemassa, mutta se ei yleensä tee mitään.

Rivit 45–83: Näkymien ja käyttöliittymän (eli CFatsAppUi-instanssin) työnjakoa voidaan ajatella näytön eri osien kautta. Toisin sanoen näkymän vastuulla on ns. asiakasalue (*client area*), joka kuvassa 6 on taustaltaan kuvioitu. Käyttöliittymä pitää puolestaan huolen (näkymän luomisen lisäksi) niistä elementeistä, jotka sisältyvät EIK_APP_INFO-resurssiin eli tässä tapauksessa valikko, työkalupalkki ja pikanäppäin.

Luokan `CFatsAppView` toteuttamisessa suurin työ (ja hankalin) on käyttöliittymän virittäminen toimintakuntoon. Kontrollien luominen ja näyttäminen on ainakin tässä tapauksessa tehty hartiavoimin yrityksen ja erehdyksen menetelmällä. Parhaiten asian kimuranttisuutta valaissee funktion `ConstructL` koko: kommentteineen 80 riviä ja funktio ei tee juurikaan muuta kuin luo kontrollit ja osittain kytkee ne yhteen. Todellisuus ei kuitenkaan valittamalla parane, joten on syytä käydä `fatsappview.cpp` tarkemmin läpi:

Esim. 14, s. 62: `fatsappview.cpp`

Rivit 9–18: Näkymän tehtävä on luoda itseään vastaava ikkuna eli asiakasalue. Alueen maksimikoko annetaan `ConstructL`:n argumenttina. Käytännössä ei ole järkevää luoda pienempää ikkunaa, koska seurauksena on sangen hämäävä efekti: käyttöliittymä näyttäisi siltä, kuin siitä olisi leikattu osa asiakasalueesta pois.

Grafiikkakontekstiasetukset riveillä 14–16 pitävät huolen siitä, että ensinnäkin tausta on halutunlainen ja toisekseen kontrollit sulautuvat taustaan oikein. Esimerkiksi rivin 14 unohtaminen aiheuttaisi sen, että valitun numerokentän otsikon (Height kuvassa 6) kohdalle jäisi ainoastaan musta läiskä siinä vaiheessa, kun valittaisiin jokin toinen kenttä.

Rivit 23–38: `CEikFloatingPointEditor`-kontrollin (paino-, pituus- ja indeksikentät) erityispiirre on se, että se hyväksyy ainoastaan laillisia desimaalilukuja,joiden pitää vielä sopia annetulle välille. Mikäli käyttäjä yrittää kirjoittaa laittoman syötteen, näytölle ilmaantuu asiasta huomauttava ikkuna. Mielenkiintoinen suunnitteluratkaisu on se, että mainittu virheentarkastelu tapahtuu vasta siinä vaiheessa, kun kontrollin jälkeen ollaan valitsemassa jokin toinen. Eli virheentarkastelu sijaitsee kontrollin `PrepareForFocusLossL`-jäsenfunktiossa.

Rivit 44–65: Näkymän käyttöliittymä koostuu kolmesta rivistä ja yksittäinen rivi taas on `CEikCaptionedControl`-instanssi, joka sisältää otsikon (*caption*), kontrollin (yllä mainitut numerokentät) ja mahdollisen jälkitekstin (*trailer*). Otsikot on kerrottu resurssitiedostossa ja vastaavat määrittelyt ovat näkyvissä myös lähdekoodissa, koska `rcomp`in tuottama `fats.rsg`-otsikkotiedosto on includoitu `fats.h`:ssa.

Mielenkiintoinen yksityiskohta on se, että indeksikentän laillisuutta ei tarkasteta, koska se ei voi tulla valituksi, sillä sen `SetNonFocusing`-jäsenfunktiota kutsutaan rivillä 65. Tästä syystä indeksikenttä on myös ainoa, jonka `SetObserver`-jäsenfunktiota ei kutsuta. Tarkkailijan asettaminen mahdollistaa tekstikentän valitsemisen hiirellä.

Rivit 68–80: Yllä mainitut rivit talletetaan `CEikCapCArray`-taulukkoon. Taulukon kooksi määrätään sen itse laskema minimikoko ja sijainniksi vasen yläkulma.

Rivit 83–85: Ohjelmoijan vastuulla on pitää huolta siitä, että haluttu kontrolli tulee kulloinkin valituksi. Tähän liittyvä logiikka haukkaakin suuren osan toteutuksesta: jäsenfunktiot `MoveFocus`, `StepFocus`, `HandleControlEventL` ja

`OfferKeyEventL` kukin tahollaan ovat valintamekanismin osia. Jäsenmuuttuja `iCurrentLine` sisältää valitun `iLines`-taulukon alkion indeksin. Alussa valittuna on ensimmäinen eli painorivi. Alkuvalinnan (rivi 85) jälkeen on aina täsmälleen yksi rivi valittuna.

Rivi 87: Näkymä on valmis ja se voidaan aktivoida, jonka jälkeen se tulee näkyviin.

Rivit 93–98: Purkajan tehtävänä on vapauttaa olion dynaamisesti varaama data. Olioiden välisiin omistussuhteisiin liittyvä mielenkiintoinen yksityiskohta on se, että `iLines` tuhoutuessaan vapauttaa myös siihen liittyvät kontrollit. Aiemmin luvussa 3.3 mainitut muistivuodot vaanivat luonnollisesti EIKON-sovelluksiakin. Huolestuneisuus kasvaa entisestään, kun huomataan, että `ConstructL`:ssä ei käytetä siivouspinoja lainkaan. Tässä yksittäisessä tapauksessa siivouspinojen puute perustellaan toisaalta sillä, että jos muistinvaraus olion alustamisen yhteydessä epäonnistuu, sovellus joutaakin kaatua ja toisaalta siivouspinon `PushL`- ja `Pop`-operaatioiden lomittaminen selkeällä tavalla voi osoittautua hyvinkin hankalaksi. `ConstructL`:ssä rakennetaan toisiinsa kiinteästi liittyviä olioita, joiden keskinäisiä omistussuhteita ei löydy dokumentaatiosta.

Rivit 102–104: `Draw`-jäsenfunktio tulee toteuttaa siten, että se ei voi aiheuttaa poikkeusta ja palaa nopeasti. Painoindeksin tapauksessa riittää pelkkä taustan piirto, koska muut kontrollit osaavat piirtää tarvittaessa itsensä.

Rivit 108–116: Näkymä toimii samalla myös alustakontrollina, joten sen täytyy pyydettäessä kertoa, kuinka monta kontrollia se sisältää ja tarjota mahdollisuus niiden indeksointiin. Painoindeksin tapauksessa on toimittu niin, että `CFatsAppView`-instanssi näyttäytyy ulospäin kolmena rivinä.

Rivit 121–127: `ValidateCurrent` kertoo, onko valitun rivin sisältämän numerokentän sisältö laillinen.

Rivit 131–139: Uuden rivin valinta edellyttäen, että nykyisen rivin numerokentän sisältö on laillinen.

Rivit 143–175: Kursorin siirtäminen numerokentästä toiseen onnistuu sekä hiirellä, jolloin kutsutaan `HandleControlEventL`-jäsenfunktiota, että näppäimistöllä. `OfferKeyEventL`:n tehtäviin kuuluu kaikkien näppäintapahtumien käsittely eli kursorin (valinnan) siirtämisen (tabulaattori, ylä- ja alanuoli) lisäksi myös painoindeksin päivittäminen (rivinvaihto). Mikäli painettu näppäin ei ole mikään näistä erityistapauksista, näppäintapahtuma välitetään eteenpäin valittuna olevalle numerokentälle.

Rivit 179–187: Vihdoinkin laskentaa! `CEikFloatingPointEditor`ista saadaan sen sisältämä arvo suoraan ulos jäsenfunktiolla `Value` ja sisään vastaavasti `SetValue`lla. Kaikkein ensimmäiseksi kuitenkin varmistutaan, että valittuna olevan numerokentän arvo on laillinen. Toisesta numerokentästä ei tarvitse

murehtia, koska sen alkuarvo on ollut laillinen (184 tai 100) ja mahdollisten muutostenkin laillisuus on tarkastettu viimeistään siinä vaiheessa, kun nykyinen kenttä on valittu.

Lopuksi piirretään indeksikenttä uudelleen, että muutos saadaan näkyviin.

Rivit 194–206: Näppäimillä voidaan hyppiä riviltä toiselle, joko alaspäin (tabulaattori ja nuoli) tai ylöspäin (nuoli). Jossain vaiheessa voidaan mennä ympäri, ja toisaalta indeksikenttä ei saa tulla valituksi. Kaiken tämän logiikan hoitaa jäsenfunktio `StepFocus` ja vieläpä siten, että rivejä voisi olla useampikin kuin tämänhetkiset kolme ja samaten myöskään valintakelvottomien rivien lukumäärää ei ole rajattu. Argumenttinaan `StepFocus` saa tiedon siitä, kumpaan suuntaan ollaan menossa. `TDir` on luokan sisäinen luettelotyyppi (ks. esim. 12).

5 Todellinen sovellus

Edellisen luvun painoindeksisovellus antaa jossain määrin harhaan johtavan kuvan EPOC-sovelluskehityksestä. Toisin sanoen esimerkin antamilla eväillä ei pääse kovinkaan pitkälle silloin, kun tarkoituksena olisi rakentaa monimutkaisempi ja tietyssä mielessä ”todellisempi” sovellus. Tällöin ensinnäkin sovelluksen pitäisi olla arkkitehtuuriltaan selkeämpi ja puhdaslinjaisempi. Painoindeksiesimerkissähän sekä toiminnallisuus (indeksin laskeminen) että käyttöliittymä oli pakattu saman luokan sisään. Toisekseen EPOC-sovellusten tiedostonkäsittelyyn liittyy omat kommervenkkinsä, joihin ohjelmoija varmasti törmää paitsi painoindeksisovelluksessa, joka ei käytä tiedostoja.

Tämä luku esittelee kokonaisen, todellisen EPOC-sovelluksen; Buzz Bingon. Nimensä mukaisesti kyseessä on tutun bingopelin variantti. Siinä, missä normaalisti bingossa on numeroita, niin Buzz Bingo sisältää sanoja tai termejä. Nimi juontaa juurensa paperiversiona leviävästä Buzzword Bingosta, joka ironisoi liikeelämän konsulttien kielenkäyttöön pesiytyneitä muoti- ja lainasanoja. Buzz Bingossa käyttäjä voi valita sanaston (tekstitiedosto), josta termit kulloinkin arvotaan. Oletussanastona on TTKK:n Ohjelmistotekniikan laitoksen luentobingoa[1] varten ideoitu sanasto (mm. framework, pattern, formaali, robusti, konventio, elegantti, aspekti, relaatio, triviaali, . . .)

EPOC-terminologiassa Buzz Bingon kaltaisia sovelluksia kutsutaan tiedostopohjaisiksi (*file-based*). Buzz Bingon tapauksessa tämä tarkoittaa seuraavaa:

- Sovellus tukee tiedostojen tallettamista ja lukemista. Tässähän ei sinällään ole mitään ihmeellistä, mutta EPOCin filosofian mukainen kytkös sovelluksen ja sen käsittelemien tiedostojen välillä on totuttua tiukempi. Sovellus tietää kunakin ajanhetkenä, minkä nimistä tiedostoa (bingolappua) se käsittelee. Tarvittaessa luodaan automaattisesti uusi tiedosto. Näin tapahtuu esimerkiksi silloin, kun sovellus käynnistetään (Extras-valikosta) ensimmäistä kertaa: uusi tiedosto on nimeltään Buzz Bingo.

  Bingolapun tallettaminen tapahtuu (käyttäjän kannalta) automaattisesti silloin, kun sovellus suljetaan tai avataan uusi tiedosto.

- Bingolapputiedoston valitseminen aiheuttaa sovelluksen käynnistymisen. Mikäli sovellus on ennestään käynnissä, käytetään tiedostonvaihtomekanismia (*file switching*): paraikaa käsiteltävä tiedosto (eli pelattavana ollut bingolappu) talletetaan ja avataan valittu tiedosto. Näin ei tosin aina tapahdu, koska EPOC-laitteet (esim. Psion 5mx) voidaan konfiguroida myös siten, että uusi tiedosto käynnistää myös uuden sovellusprosessin.

- Käyttäjä voi luoda uusia Buzz Bingo -tyyppisiä dokumentteja Systemsovelluksen New file -napilla.

[1] `http://www.cs.tut.fi/~vespe/bingo`

- Buzz Bingo -pelilappuja voidaan upottaa osaksi muiden sovelluksien dokumentteja (esim. Word).

Buzz Bingon lähdekoodi löytyy liitteestä B alkaen sivulta 66 pois lukien ne osat, jotka on sisällytetty tähän lukuun. Lähdekoodin ja tehtyjen ratkaisujen läpikäynti aloitetaan arkkitehtuuritasolta, jonka jälkeen siirrytään tiedostonkäsittelyn kautta käyttöliittymäpuolelle. Lopuksi kerrotaan, miten sovelluksen asentaminen EPOC-laitteeseen tapahtuu.

5.1 Arkkitehtuuri

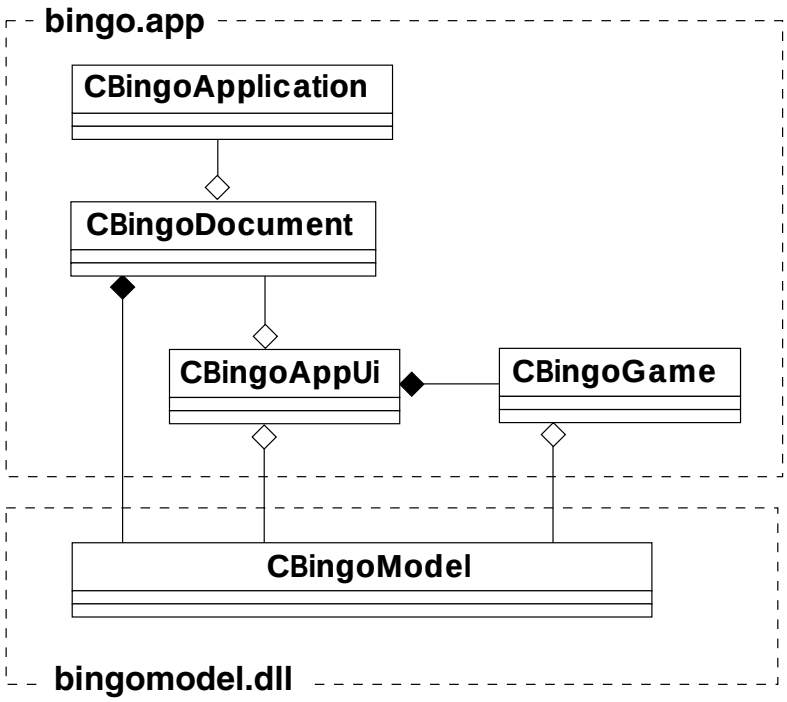
Karkeimmalla tasolla Buzz Bingo koostuu kahdesta erillisestä osasta: dynaamisesta kirjastosta `bingomodel.dll` ja EIKON-sovelluksesta `bingo.app`. Tämäntyyppinen kahtiajako on sekä käytännöllistä että EPOCin MVC-filosofian mukaista (ks. luku 2.1). Erillistä, käyttöliittymästä riippumatonta kirjastoa on helpompi koodata ja testata kuin kokonaista EIKON-sovellusta. Käytännössä tämä tarkoitti sitä, että `bingomodel.dll`:n testaaminen tapahtui samantyyppisellä konsolisovelluksella kuin esimerkiksi aikaisemmat deskriptori- ja siivouspinoesimerkit.

Eri osien (eli varsinaisen sovelluksen ja kirjaston) sisältämät luokat näkyvät kuvassa 7: `bingomodel.dll` koostuu ainoastaan luokasta `CBingoModel`, joka mallintaa yhtä bingolappua. Sovellusluokkia ovat ”pakolliset” `CEikApplication`-, `CEikDocument`- ja `CEikAppUi`-erikoistukset. `CBingoGame` toimii käyttöliittymänäkymän roolissa.

Kuvassa on korostettu erityisesti luokkien välisiä omistussuhteita. On tärkeää huomata, että vaikka kirjaston sisältämään `CBingoModel`-luokkaan on viittauksia usealta eri suunnalta, sen omistaja on sovelluksen ”dokumentti” eli luokasta `CEikDocument` periytetty `CBingoDocument`. Ohjelman ajoaikaisen rakenteen kannalta tämä tarkoittaa sitä, että bingolappujen luominen ja tuhoaminen on dokumentin vastuulla.

MVC:n kannalta kirjasto (eli luokka `CBingoModel`) vastaa mallia (tai ainakin sen keskeistä osaa). EPOC-terminologiassa tämäntyyppistä kirjastoa kutsutaan usein termillä *engine*. Käyttöliittymäpuolella mallia vastaa luonnollisesti dokumentti eli `CBingoDocument`-instanssi. Loput käyttöliittymäluokat voidaan tulkita osin kontrollereiksi ja osin näkymiksi. Selkeästi `CBingoGame` on enemmän näkymän kaltainen ja `CBingoAppUi` kontrolleri mutta varsinaista selkeää rajaa ei ole, sillä toisaalta `CBingoGame`-instanssi käsittelee mallia suoraan ja `CBingoAppUi`n tehtäviin kuuluu mm. valikoiden, dialogien ja ääniefektien hallinta.

Luokan `CBingoModel` toteutus on pohjimmiltaan hyvin yksinkertainen. Bingolappu mallinnetaan vektorina `CCell`-olioita, joista kukin tietää mitä termiä se edustaa ja onko se valittu (eli kyseistä termiä on käytetty esimerkiksi luennolla) vai ei. Tutkittaessa kirjaston esittelyä ja toteutusta (esimerkit 16 ja 17) törmätään äkki-



Kuva 7. Bingoluokat.

seltään hämäävään käytäntöön: kirjaston julkiseen rajapintaan kuuluvien funktioiden tarkennin on esittelyssä `IMPORT_C` ja toteutuksessa `EXPORT_C`. Painoindeksiesimerkistä muistetaan, että `EXPORT_C` kertoo funktion löytyvän DLL:stä. `IMPORT_C` välittää saman tiedon kääntäjälle, joten käännös onnistuu vaikka kirjasto ladataankin vasta ajoaikana. Kirjastoluokan yksityinen rajapinta ei tarkentimia kaipaa, koska asiakaskoodi eli kirjaston käyttäjä ei pääse siihen käsiksi.

5.2 Tiedostot, streamit ja storet

Datan tallentaminen ja lukeminen voi tuntua EPOCissa aluksi sangen hankalalta. Syynä on ensinnäkin tiedostonkäsittelyn originelli ja siten äkkiseltään outo toteutus ja toisaalta sovellusten erikoinen tiedostopohjaisuus, joka mainittiin edellisessä luvussa. Tämä ei kuitenkaan tarkoita sitä, että ohjelmoija olisi tyystin mahdottoman tehtävän edessä, sillä apuna on kerrankin sangen kattava SDK:n mukana tuleva dokumentaatio ja hyvät esimerkit. Erityisesti SDK:n ns. tutoriaaliosuus eli Boss Puzzle -sovelluksen eri variaatiot on korvaamattomana apuna tutustuttaessa EPOC-arkkitehtuurin syvimpään olemukseen. Itse asiassa esimerkit ovat tältä osin niin hyviä, että ymmärtämättömästi kopioimallakin saa lähestulkoon toimivan sovelluksen.

Käydään ensin lyhyehkösti läpi EPOCin tiedostonkäsittelyn erikoispiirteet, minkä jälkeen tutkitaan, miten ne realisoituvat Buzz Bingossa.

EPOCin tiedostojärjestelmä(kin) pohjautuu asiakas-palvelinmalliin, kuten jo luvussa 2.1 lyhyesti todettiin. Toisin sanoen, vaikka varsinaista fyysistä levyä ei

EPOC-laitteessa olisikaan[2], tiedostopalvelimen (*File Server*) kautta saadaan käytössä olevaan muistiin tutunkaltainen näkymä eli Windowsista tuttu VFAT-tiedostojärjestelmä. Tämänkaltainen tiedostojärjestelmä on hyvä olla yksinomaan senkin takia, että tiedostojen siirto pöytä- ja EPOC-koneen välillä onnistuisi mahdollisimman kivuttomasti. Palvelimiin perustuva malli takaa myös laajennettavuuden esimerkiksi muiden tiedostojärjestelmien suuntaan. `CBingoModel`-luokan `ConstructL`-jäsenfunktio on hyvä esimerkki tiedostopalvelimen käytöstä:

Esim. 17, s. 68:
`bingomodel.cpp`

Rivit 17–24: EPOC-palvelinprosesseihin otettua yhteyttä kuvataan istuntooliolla (luokan `RSessionBase` lapsiluokan instanssi). Tiedostopalvelimen tapauksessa istunnon tyyppi on `RFs` ja sen jäsenfunktio `Connect` muodostaa yhteyden. Samassa istunnossa voidaan käsitellä useampaakin tiedostoa ja yksittäinen tiedosto kuvautuu `RFile`-instanssiksi. Tiedostoa avattaessa pitää istunnon lisäksi antaa tiedoston nimi ja avaustila. Nimi voi olla absoluuttinen tai suhteessa istunnon työhakemistoon (*session path*). Avaustila vaikuttaa varsinaisen perustilan (luku, kirjoitus) lisäksi myös tiedoston lukituksiin eli siihen, pääsevätkö muut prosessit yhtäaikaa käsiksi samaan tiedostoon. Tässä tapauksessa tiedosto on kaikkien käytössä mutta ainoastaan lukemista varten.

Rivit 25–37: Sanatiedosto on tavallinen tekstitiedosto, jossa yksittäinen rivi vastaa yhtä sanaston termiä. Ennen tämän nimenomaisen bingolapun sanojen arpomista täytyy koko sanasto lukea yhteen tietorakenteeseen. Kyseinen tietorakenne `words` on dynaaminen (osoitin)vektori, jonka kukin alkio on tyyppiä `HBufC*` ja jonka alkutilavuus on 10 alkiota.

Rivit 68–80: Luokka `TFileText` on eräänlainen tiedostoiteraattori, jolla voidaan lukea tiedosto rivi kerrallaan (edellyttäen, että rivin pituus ei ylitä 256 merkkiä). Varsinainen lukeminen tapahtuu silmukassa riveillä 74–79: kullakin kierroksella tiedostoiteraattori kopioi lukemansa rivin puskurimerkkijonoon `buf`. Tämän jälkeen luodaan puskurista dynaaminen kopio `term` (tyyppiä `HBufC*`) ja lisätään se vektoriin `words`. Näin jatketaan, kunnes lukeminen epäonnistuu eli on (luultavasti) päästy tiedoston loppuun.

Miksi yllä mainittu silmukka on omana – yksinkertaisena – funktionaan `ReadWordsL` eikä esimerkiksi `ConstructL`:n osana? Syynä on virheenkäsittely ja tarkemmin vielä muistivuotojen estäminen, joka on EPOCissa ensiarvoisen tärkeää, sillä jos sovellus ei sulkeuduttuaan ole vapauttanut kaikkea varaamaansa muistia, huomauttaa ajoympäristö siitä joko ”kaatumalla” mystiseen virheilmoitukseen (emulaattorin debug-tila) tai panic-ilmoituksella (emulaattorin release-tila tai EPOC-laite). Muistivuotojen pääasiallisena estomekanismina EPOCissa ovat luvusta 3.3 tutut TRAP ja siivouspino. Pääperiaatteena on, että dynaamisesti varatut paikalliset oliot

[2]Ei se tietenkään mahdotontakaan ole. Esimerkiksi Psion netBookin [5] Compact Flash -korttipaikka on IBM Microdrive -yhteensopiva [2].

talletetaan siivouspinoon, josta ne poistetaan joko käyttöjärjestelmän avulla (poikkeustilanteissa eli jos funktiosta poistutaan leave-mekanismilla) tai eksplisiittisesti normaalitilanteessa. Tästä syystä `words` on pistetty siivouspinoon heti luomisensa jälkeen. Ikävä kyllä `CArrayPtrFlat` on toteutettu niin, että sen purkaja ei automaattisesti tuhoa dynaamisesti varattuja alkioita. Tämä on toki siinä mielessä ymmärrettävää, että alkiothan voivat joissain tapauksissa olla myös staattisia. Dynaamisten alkioiden tapauksessa vektori pitää eksplisiittisesti tyhjentää `ResetAndDestroy`-kutsulla, joten ohjelmoijan täytyy tältä osin varautua myös poikkeustilanteisiin: muistin loppuminen aiheuttaa poikkeuksen `ReadWordsL`:ssä, joka on vuorostaan TRAPin sisällä (rivi 29). Virhetilanteissa (muisti loppuu tai sanoja liian vähän) tyhjennetään `words`, ja heitetään poikkeus ylöspäin (rivit 32–37).

Rivit 38–52: `ConstructL`:n tiedostonkäsittelyosuus on tässä vaiheessa jo tehty. Jäljellä on enää bingolapun luonti. Bingolappu mallinnetaan vektorina, jonka alkiot ovat tyyppiä `CBingoModel::CCell*`. Jäsenmuuttuja `iDim`, joka kertoo lapun rivien tai sarakkeiden lukumäärän, päivitetään vasta vektorin tilanvarauksen jälkeen. Jos tilanvaraus epäonnistuu ja aiheutuu poikkeus, `iDim`in arvo on nolla, kuten intuitiivista onkin. Vektorin ”nollaus” riveillä 46–48 on tarpeellista vastaavasta syystä: vektorin alkioita vastaavat `CCell`-instanssit luodaan dynaamisesti. Koska kyseessä on luokan jäsenmuuttuja `iGrid`, alkioita ei tarvitse pistää siivouspinoon, sillä niihin päästään käsiksi luokan `CBingoModel`-purkajassa (rivit 83–90). Entäpä jos muisti loppuu kesken alkioiden varaamisen (silmukka riveillä 50–52)? Tällöinhän purkajassa yritettäisiin vapauttaa myös niitä olemattomia alkioita vastaavia muistialueita. Nollaamisen ansiosta tämä ei ole ongelma, sillä nollaosoittimen ”vapauttaminen” on sallittua ja hyväksyttävää.

Rivit 53–63: Jäljellä on enää bingolapun termien arvonta. Käytettyjen tietorakenteiden (`words` ja `iGrid`) kannalta tämä tarkoittaa sitä, että arvottujen `words`in alkioiden (merkkijonojen) omistus siirtyy `iGrid`-alkioiden (eli `CCell`-instanssien) vastuulle.

`Math::Rand`-funktio palauttaa satunnaisluvun. Argumenttina annetaan siemenluku, joten sen tulisi pysyä samana arvontojen välillä. Koska kyseessä ei kuitenkaan ole mitenkään yltiökriittinen järjestelmä, satunnaisuuden ”hyvyyteen” ei olla kiinnitetty sen kummemmin huomiota. Toisin sanoen siemenluku (eli ”tiksujen” lukumäärä) pysyy samana vain yhden `CBingoModel`-instanssin luomisen ajan ja lisäksi skaalaus halutulle välille tehdään suruttoman brutaalisti ottamatta huomioon jakauman mahdollista vinoutumista.

Aivan lopuksi tyhjennetään `words` eli ylijäämäsanat ja tuhotaan tarpeettomaksi käynyt tietorakenne.

`ConstructL`:n tapauksessa luettavan sanatiedoston formaattina on tavallinen tekstitiedosto siirrettävyyssyistä: samaa sanastoa voidaan käyttää useammassa bingototeutuksessa eri ympäristöissä. Tapauksiin, joissa tiedoston siirrettävyydel-

lä ei ole merkitystä, tarjoaa EPOC käytettäväksi vahvat rohdot eli streamit, storet ja tuen sarjallistamiselle.

Stream tarkoittaa yleiskäsitteenä EPOCissa samaa kuin syöttö- ja tulostustoiminnoissa yleensä eli tietovuota, johon voidaan kirjoittaa ja/tai josta voidaan lukea dataa. Kooditasolla kukin stream on pohjimmiltaan periytetty abstrakteista kantaluokista `RReadStream` ja `RWriteStream`. Toisin sanoen streamien käyttökelpoisuus ei ole mitenkään sidottu esimerkiksi tiedostoihin, vaan pikemminkin päinvastoin: EPOCissa streamit toimivat yleispätevänä datan siirtomuotona. Sarjallistamismekanismin avulla olio voidaan kirjoittaa tiloineen kaikkineen streamiin ja myöhemmin uusi olio voidaan palauttaa tähän tilaan. Tämä tapahtuu vieläpä tutuilla stream-operaattoreilla << ja >>. Kaikki oliot eivät kuitenkaan oletuksena sarjallistu vaan ainoastaan ne, joiden toteutuksessa on otettu tämä mahdollisuus huomioon eli ne ovat ”stream-tietoisia” (*stream-aware*) olioita. `CBingoModel`in sarjallistuminen näkyy siinä, että sen rajapinnasta löytyvät jäsenfunktiot `ExternalizeL` ja `InternalizeL`:

Rivit 176–182 ja 274–277: Perustyyppien (tai niiden EPOC-vastineiden) kirjoittamiselle streamiin löytyy omat funktionsa. Olioiden kirjoittaminen tapahtuu operaattorilla <<, edellyttäen että kyseinen luokka määrittelee `ExternalizeL`-funktion, kuten tässä tapauksessa (rivi 180) `CCell` (ja tietysti myös `CBingoModel`). Mekanismin käyttökelpoisuutta ja ilmaisuvoimaa kuvaa se seikka, että hyvin pienellä koodimäärällä saadaan toimiva sarjallistaminen toteutettua: kukin luokka hoitaa oman sarjallistumisensa, jolloin koodin rakennekin säilyy mukavan kerrosmaisena.

Rivit 186–196 ja 281–284: Lukeminen tapahtuu vastaavalla tavalla kuin kirjoittaminenkin. Asiakaskoodin vastuulle jää muistivuotojen estäminen: `CBingoModel`-instanssin tulee olla ”tyhjä” ennen lukemista.

Store on stream-säiliö, joka koostuu yhdestä tai useammasta streamistä. Tietyssä mielessä store on EPOCin vastine tiedostolle: sovellusten käsittelemä data talletetaan yleensä nimenomaan storeihin ja aiemmin mainittu sovellusten tiedostopohjaisuus tarkoittaa itse asiassa store-pohjaisuutta:

- Kytkös UID:n ja sovelluksen generoiman streamin välille tehdään storessa. Toisin sanoen store voi sisältää ns. stream-luettelon (*stream dictionary*), jossa kerrotaan kutakin storen sisältämää streamiä vastaava UID eli sovellus.
- EIKON-sovellus tietää koko ajan, minkä storen kanssa se on tekemisissä eli mitä storea käyttöliittymän näkymät esittävät. Tähän päästoreen (`MainStore`) päästään käsiksi sovellusta vastaavan `CEikProcess`-instanssin kautta. Lisäksi luokka `CEikAppUi` sisältää valmiit mekanismit storen päivittämiseen (jäsenfunktiot `SaveAnyChangesL` ja `SaveL`).

- Streamien lisäksi store voi sisältää myös muita, upotettuja storeja (*embedded store*). Sovellusten kannalta tämä tarkoittaa sitä, että esimerkiksi tekstinkäsittelydokumenttiin voidaan upottaa jollain muulla sovelluksella tehty osio, esimerkiksi kaavio. Koska tuki tällaiselle mekanismille tarjotaan jo käyttöjärjestelmätasolla, ei ohjelmoijan tarvitse käytännössä nähdä lainkaan ylimääräistä vaivaa upotettavan sovelluksen tekemiseen.

Kooditasolla storen käsittely on hyvin suoraviivaista: siinä missä olioiden sarjallistamismekanismi perustui jäsenfunktioihin `ExternalizeL` ja `InternalizeL`, storeja tukevien (*store-aware*) luokkien tulisi toteuttaa jäsenfunktiot `StoreL` ja `RestoreL`. Esimerkiksi luokan `CBingoModel` toteutukset:

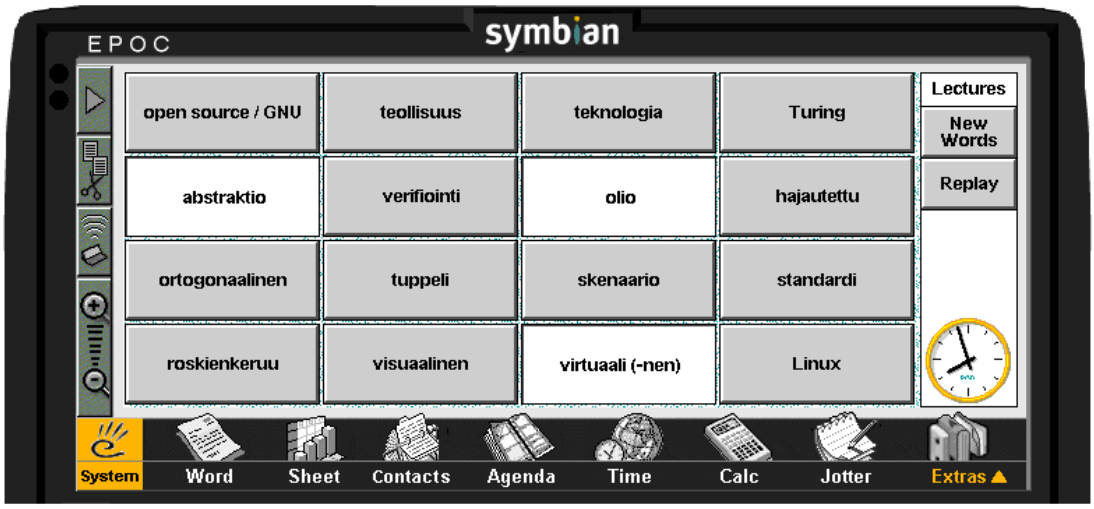
Rivit 244–251: `StoreL` saa argumentikseen viitteen (jo avattuun) `CStreamStore`-instanssiin. Stream-luettelon tekeminen tapahtuu ylemmällä tasolla (tarkemmin sanottuna `CBingoDocument`-luokan `StoreL`-funktiossa), joten funktion täytyy palauttaa luomansa streamin tunniste. Store-mekanismi tukee jonkin verran transaktion käsitettä: storen sisällön muutokset astuvat voimaan vasta `CommitL`-kutsun jälkeen.

Rivit 255–261: Storesta luettaessa streamin tunniste annetaan argumenttina. Tässä tapauksessa tosin storessa ei ikinä olekaan kuin yksi stream.

Lienee syytä mainita, että `CBingoModel`in rajapinta sisältää myös sellaisia tiedostonkäsittelyfunktioita kuten `SaveL`, jotka ovat lopullisen sovelluksen kannalta tarpeettomia mutta joita voidaan käyttää testitarkoituksiin silloin, kun ylempiä kerroksia ei ole eli esimerkiksi tekstipohjaisten sovellusten tapauksessa.

Edellä käsiteltiin nimenomaan sovelluksen alimman tason tiedostonkäsittelyä. Sovelluksen arkkitehtuurin ja EPOCin kannalta yleensäkin on hyödyllistä tutkailla ylempien kerrosten työnjakoa. Etenkin tiedostopohjaisuuden vaatimusten heijastuminen kooditasolla on syytä käydä läpi. Tiedostonkäsittelynkin suhteen sovelluksen arkkitehtuuri pyrkii noudattamaan MVC-mallia kuvan 7 (s. 43) esittämällä tavalla: pääasiallisena vastuuoliona, jonka tehtäviin kuuluu tiedostojen luominen ja lukeminen, toimii `CBingoDocument`. Koska dokumentti on EIKON-sovellusten keskeinen käsite ja `CBingoDocument` noudattaa store-protokollaa eli tarjoaa soveliaat `StoreL`- ja `RestoreL`-toteutukset, saa sovellus monet tiedostonkäsittelyyn liittyvät mekanismit valmiina. Samalla tietysti tämäntyyppinen ”implisiittisyys” vaikeuttaa sovelluksen suoritusmallin ymmärtämistä. Kuten aiemmin todettiin, tilannetta helpottaa EPOCin tältä osin aika kattava dokumentaatio ja etenkin hyvät esimerkit.

EPOCin tiedostonvaihtomekanismi olettaa, että `CBingoAppUi` tarjoaa jäsenfunktiot `OpenFileL` ja `CreateFileL`. Näin ainakin dokumentaatio väittää mutta esimerkiksi debuggerin mukaan `CreateFileL`-funktiota ei kutsuta, kun EPOC luo uuden Buzz Bingo -tyyppisen tiedoston.



Kuva 8. Buzz Bingon käyttöliittymä emulaattorissa.

5.3 Käyttöliittymä

Buzz Bingon käyttöliittymä, joka näkyy kuvassa 8, noudattaa toteutukseltaan luvussa 4.2 tutuksi tulleita perusperiaatteita: valikkojen ja palkkien käsittely on `CBingoAppUin` vastuulla, ja varsinaisen pelialueen eli bingolapun nappien esittämiseen liittyvät seikat ovat osa erillistä näkymää eli `CBingoGame`-instanssia.

Kuvassa näkyvän toiminnallisuuden (arvotaan nappeihin uudet sanat, uusi peli samoilla sanoilla, sanojen valinta eli nappien painelu) lisäksi käyttäjä voi Bingo-valikon avulla käsitellä bingolappuja (luonti, talletus, avaaminen), lukea turhamaisen About-ikkunan ja poistua sovelluksesta. Yksittäinen bingolappu vastaa aina yhtä tiedostoa ja tiedoston nimi näkyy EPOC-käytännön mukaisesti oikeassa yläkulmassa.

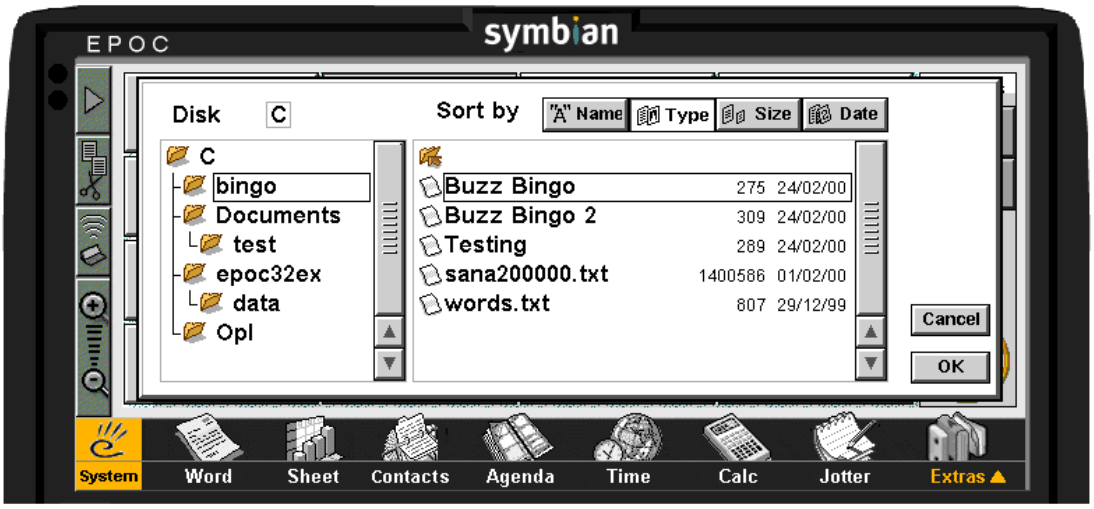
Näkymän koodissa ei sinällään ole mitään erityisen uutta tai ihmeellistä aikaisempaan painoindeksiesimerkkiin verrattuna, joten tarkastellaan Buzz Bingon eksoottisempia piirteitä; dialogeja ja ääntä.

Dialogityypistä riippumatta dialogin käyttö koostuu seuraavista vaiheista:

1. Dialogi alustetaan eli `CEikDialog`-instanssi luodaan ja parametrisoidaan.

2. Dialogi näytetään eli kutsutaan dialogin jäsenfunktiota `ExecuteLD`. Mielenkiintoinen ja jossain mielessä myös omituinen yksityiskohta EPOCin nimeämiskäytännössä on D-kirjaimen merkitys: dialogi-instanssi tuhotaan kutsun jälkeen! `ExecuteLD`:lle annetaan argumentiksi dialogin ulkoasun määrittelevä resurssitunniste.

3. `ExecuteLD`:n paluuarvon analysointi tarvittaessa eli käytännössä silloin, kun dialogi voidaan sulkea useammasta napista (esim. OK ja Cancel).



Kuva 9. EIKONin tiedostoselain.

Kuten olettaa sopii, EIKON tarjoaa valmiit (`CEikDialogista` periytetyt) dialogitoteutukset yleisimpiin tarkoituksiin. Näistä Buzz Bingo käyttää seuraavia:

`CEikInfoDialog`. Perusdialogi, joka sisältää Continue-napin. Dialogin luontivaiheessa sille kerrotaan varsinaisen ”leipätekstin” lisäksi ikkunan otsikko. Buzz Bingossa `CEikInfoDialogia` käytetään `CBingoAppUin` jäsenfunktioissa `ErrorL`, `AboutL` ja `BingoL`.

Esim. 24, s. 83: `bingoappui.cpp`

`CEikFileOpenDialog`. Dialogi tarjoaa kaksitasoisen näkymän allaolevaan tiedostojärjestelmään: tiedoston nimi-, hakemisto- ja asemakenttien lisäksi Browse-napista voi käynnistää erityisen tiedostoselaimen (*file browser*), joka näkyy kuvassa 9. Buzz Bingo käyttää dialogia talletettujen bingolappujen avaamiseen jäsenfunktiossa `OpenBingoL` (rivit 184–197). `RestrictToNativeDocumentFiles`-kutsu rivillä 190 rajoittaa ensimmäistä dialoginäkymää (eli ei-selainta) siten, että valittavissa on ainoastaan Buzz Bingon luomat tiedostot. Tiedoston tyyppi on sama kuin sen sisältämä UID. `ExecuteLD` evaluoituu todeksi, kun OK-nappia painetaan. Valitun tiedoston nimi on talletettu dialogin alustuksen yhteydessä annettuun `TFileName`-instanssiin.

`CEikFileSaveAsDialog`. Tallennusdialogia tarvitaan ainoastaan silloin, kun bingolappu halutaan tallettaa uudella nimellä (käyttöliittymän Bingo-valikon valinta Save As…ja vastaava jäsenfunktio `SaveAsBingoL` riveillä 248–274).

Oman dialogitoteutuksen rakentaminen tapahtuu seuraavasti:

1. Sovelluksen resurssitiedostossa kuvataan dialogin ulkoasu. `DIALOG`-resurssi

Esim. 19, s. 74: `bingo.rss`

Kuva 10. Dialogi uuden bingolapun luomiseen.

(rivit 120–159) sisältää yleisattribuuttien (perusnapit, otsikko ja modaalisuus) lisäksi listan `DLG_LINE`-resursseja. Toisin sanoen dialogi voidaan koostaa selite-kontrolli -pareista, joista kukin vastaa yhtä dialogin ”riviä”. Esimerkiksi riveillä 125–132 luodaan pari, jonka selite on Dimension, ja varsinaisena kontrollina on (painoindeksistä tuttu) numeroeditori-instanssi, jolla voidaan valita kokonaisluku väliltä 1–10. Syntyneen dialogin ulkoasu näkyy kuvassa 10.

Sovelluksen toiminnan (eli tarkemmin luokan `CBingoAppUi`) kannalta olennaista on, että kullakin dialogin kontrollilla on yksilöllinen (`bingo.hrh`:ssa määritelty) tunniste. Sovelluskoodissa päästään käsiksi kontrolliin tunnisteen avulla, jolloin ulkoasua ja esim. selitetekstejä voidaan muuttaa varsinaiseen sovelluslogiikkaan kajoamatta.

Esim. 24, s. 83: `bingoappui.cpp`

2. Periytetään luokasta `CEikDialog` oma dialogitoteutus eli Buzz Bingon tapauksessa luokka `CBingoAppUi::CNewBingoDialog`. Dialogien erikoistamisrajapintaa `CNewBingoDialog` kuormittaa seuraavasti:

   Rivit 305–318: Juuri ennen dialogin näyttämistä kutsutaan `PreLayoutDynInitL`-jäsenfunktiota. Tämä tarjoaa mahdollisuuden alustaa dialogin kentät ajoaikaisesti sopivilla arvoilla. `CNewBingoDialogin` toteutus täyttää kentät senhetkisen, ”vanhan” bingolapun arvoilla. Dimensiokenttä täytetään jo heti rivillä 306 mutta sanatiedostokentät vasta rivillä 317.

   Riveillä 307–310 päivitetään luokan jäsenmuuttujat osoittamaan oikeisiin kontrolleihin. Oletussanatiedosto `words.txt` sijaitsee sovelluksen ”kotihakemistossa” eli suurella todennäköisyydellä käyttäjälle oletuksena näkymättömällä systeemilevyllä. Rivillä 309 varmistetaan, että myös systeemihakemistot näkyvät käyttäjälle.

   `CNewBingoDialog` on hyvin tyypillinen tiedostodialogi siinä mielessä, että vaikka se sisältää erikseen tiedosto-, hakemisto- ja asemaeditorin,

niiden tarkoitus on kuitenkin toimia yhdessä eli kuvata samaa tiedostoa. Tällaista yhteispeliä tai editorien kytkemistä varten kontrollit tarjoavat tarkkailijamekanismin (*observer*): kontrollin sisällön muuttaminen vaikuttaa myös tarkkailijaan. Toisin sanoen rivien 313 ja 314 vaikutuksesta tiedostoeditorin päivittäminen (koko polkunimellä) muuttaa asema- ja hakemistoeditorienkin sisällöt vastaaviksi.

Rivit 323–352: `CNewBingoDialog` sisältää kolme nappia: Browse, Cancel ja OK. Peritty toteutus toimii niin, että Cancel-napin painaminen aiheuttaa dialogin sulkemisen (ja `ExecuteLD`:n paluuarvoksi tulee `EFalse`). Muiden nappien painaminen välittyy jäsenfunktiolle `OkToExitL`, jonka paluuarvo (funktion nimen mukaisesti) kertoo, voidaanko dialogi sulkea. `CNewBingoDialog` käyttää tätä mekanismia Browse-napin toiminnallisuuden toteuttamiseen eli tiedostoselaimen käynnistämiseen ja dialogin kenttien päivittämiseen selailun jälkeen. Sinällään on jonkinasteinen pettymys, että vaikka EIKON tarjoaa Browse-napin ja tiedostoselaimen valmiina, ohjelmoija joutuu silti itsekin näpräämään näinkin perustason asioiden parissa. Tiedostoselaimen eli `CEikFileBrowserDialog`-instanssin luominen ja käyttö tapahtuvat riveillä 337–345.

Mikäli painettu nappi olikin OK, rivi 350 estää dialogin sulkemisen siinä tapauksessa, että valittu tiedostonimi on laiton. Sulkemisen sijaan käyttäjää informoidaan tilanteesta oikean yläkulman ”infokuplan” avulla.

Buzz Bingon äänituki ei ole kovin kehittynyt: ääntä käytetään ainoastaan silloin, kun pelaaja saa bingon. Tällöin onnitteluikkunan avaamisen lisäksi soitetaan silmukassa äänitiedostoa `KDefaultSoundFile` (määritelty `bingo.h`:ssa). Kun bingo tulee, äänen suhteen tapahtuu seuraavaa:

Rivit 49–58: Äänitiedoston soittamiseen käytetään `CSoundPlayer`-instanssia. Luokan rakentajalle annetaan avattu ja konfiguroitu ääniajuri eli viite `RDevSound`-instanssiin. Toisena argumenttina on viite `MSoundPlayerObserver`-instanssiin, joka tässä tapauksessa on myös luokan `CBingoAppUi` instanssi. `MSoundPlayerObserver`in määrittelemä mixin-protokolla sisältää ainoastaan jäsenfunktion `PlayComplete`, jota kutsutaan automaattisesti, kun äänitiedosto on saatu soitettua.

Esim. 24, s. 83: `bingoappui.cpp`

Rivit 74–81: Äänitiedoston soittaminen tapahtuu `CSoundPlayer`in jäsenfunktiolla `PlayFileL`.

Rivit 86–90: Koska soittamisen jälkeen kutsutaan `PlayComplete`-jäsenfunktiota, toisto saadaan aikaiseksi kutsumalla `PlaySoundL`:ää uudestaan. Miksi tämä täytyy tehdä näin hankalasti? Syynä on `CSoundPlayer`in asynkroninen toteutus: soittaminen on toteutettu kulissien takana EPOCin aktiivisten olioiden (*active objects*) avulla, jotka ovat osa EPOCin rinnakkaisuusmekanismeja.

Äänitiedosto soi ikään kuin taustalla eikä vaikuta sovelluksen muuhun toimintaan.

Rivit 64–67: Dialogin sulkeuduttua keskeytetään äänitiedoston soitto ja tuhotaan vastaava `CSoundPlayer`-instanssi sekä suljetaan ääniajuri.

5.4 Paketointi

EPOC määrittelee sovelluksille erityisen asennustiedostoformaatin. Yksittäinen asennustiedosto sisältää kaiken sovellukseen liittyvän, kuten kohdelaitteen arkkitehtuurille käännetyt ohjelmatiedostot, muut tiedostot (esim. resurssit ja ikonit) ja lisäksi asennus- ja konfigurointivaihetta ohjaavat tiedostot. Käydään läpi työvaihe työvaiheelta Buzz Bingon paketointi. Kohdekoneena tässä tapauksessa on Psion 5mx.

SDK:n `makesis`-työkalu rakentaa annetun pakettitiedoston pohjalta varsinaisen asennustiedoston. Buzz Bingon pakettitiedosto `bingo.pkg` näkyy esimerkissä 8. Tiedosto lähinnä kertoo, mitkä tiedostot paketoidaan mukaan kehityskoneelta ja mihin ne puretaan kohdekoneessa. Huutomerkki korvautuu asennusvaiheessa annettavalla asematunnisteella.

Esimerkki 8. Buzz Bingon pakettitiedosto `bingo.pkg`.

```
#{"Buzz Bingo"},(0x010005ff0),1,16,0
"\vps\bingo\pkg\note.txt" - "", FT, TC
"\epoc32\release\marm\rel\bingo.app" - "!:\system\apps\bingo\bingo.app"
"\epoc32\release\marm\rel\bingo.rsc" - "!:\system\apps\bingo\bingo.rsc"
"\vps\bingo\bingoaif.aif" - "!:\system\apps\bingo\bingo.aif"
"\epoc32\release\marm\rel\bingomodel.dll" - "!:\system\apps\bingo\bingomodel.dll"
"\vps\bingo\pkg\words.txt" - "!:\system\apps\bingo\words.txt"
"\vps\bingo\pkg\bingo.snd" - "!:\system\apps\bingo\bingo.snd"
```

SDK:n mukana tulee ristiinkäännösympäristö (tarkemming `gcc`), jolla käännös kohdekoneen binääriksi onnistuu (ns. MARM build). Ohjelmoijan ei tarvitse asiasta kummemmin murehtia, vaan käännöksen perustana ovat sovelluksen `mmp`-tiedostot (esimerkit 15 s. 66 ja 18 s. 73). SDK:n `makmake`-työkalu rakentaa kunkin `mmp`-tiedoston pohjalta Visual C++:n `nmaken` ymmärtämän makefilen. Syntyneet binäärit ilmaantuvat hakemiston `\epoc32\release\marm\rel` alle yhdessä käännetyn resurssitiedoston kanssa.

Tiedosto `bingoaif.aif` on Buzz Bingon AIF-tiedosto (ks. luku 2.2 alkaen sivulta 9), joka on generoitu `aiftool`-työkalulla. `aiftool` saa argumentteinaan resurssitiedoston (esimerkki 9) ja ikonitiedoston. SDK sisältää työkalun `bmconv`, jonka avulla joukko Windows-tyyppisiä bittikarttakuvia saadaan koottua yhdeksi EPOCin ymmärtämäksi MBM-tiedostoksi (*Multi-Bitmap*).

Esimerkki 9. Buzz Bingon AIF-resurssit.

```
// bingoaif.rss
// aiftool:in kaipaama resurssitiedosto

#include <aiftool.rh>

RESOURCE AIF_DATA {
  app_uid=0x10005ff0;
  caption_list= {
    CAPTION { code=ELangEnglish; caption="Buzz Bingo"; }
  };
  num_icons = 6;
  embeddability = KAppEmbeddable;
  newfile = KAppSupportsNewFile;
}
```

AIF-resurssitiedosto tavallaan kuvailee sovelluksen piirteet ympäristölle. Tärkein on tietysti sovelluksen UID. Koska kyseessä on ”todellinen” eli periaatteessa julkiseen levitykseen tarkoitettu sovellus, on Buzz Bingolle hankitty Symbianilta omat UID:nsä (eli myös kirjastolle `bingomodel.dll` omansa). Asiaan liittyvä byrokratia on erittäin kevyt, lähes olematon: osoitteeseen *uid@epocworld.com* lähetetään viesti, josta ilmenee joko sovelluksen tekijän tai vaihtoehtoisesti sovelluksen nimi, vastuuhenkilön sähköpostiosoite ja tarvittava UID-lukumäärä (ilman erillisiä perusteluja maksimimäärä on kymmenen kappaletta). UID:t tulevat lyhyen ajan (tässä tapauksessa n. 15 minuuttia!) kuluttua sähköpostitse.

Periaatteessa EPOC-sovellus voi sisältää monikielisyystuen. Tällöin sovelluksen käyttäjälle esim. Extras-valikossa näkyvä sovelluksen nimikin voi olla eri valitusta kielestä riippuen. Resurssi `caption_list` sisältää nimen eri vaihtoehdot. Buzz Bingon tapauksessa ainoa tuettu kieli on englanti.

Koska EPOC-laitteen käyttöliittymä tukee ”zoomausta” tulee sovelluksen ikonistakin olla eri versiot eri resoluutioille. On sovelluksen tekijän asia, minkäkokoisia ikoneita hän MBM-tiedostoon pakkaa. Buzz Bingossa kokoluokkia on kuusi.

Kuten luvussa 5.2 (alkaen sivulta 43) todettiin, Buzz Bingo tukee store-mekanismia eli yksittäinen bingolappu sarjallistuu kivuttomasti ja se voidaan upottaa osaksi toisen sovelluksen storea. Samaten sovellus tukee uusien tiedostojen luomista kuvan 11 esittämällä tavalla. Nämä piirteet kerrotaan myös AIF-resurssitiedostossa.

Muutama sana itse asennustapahtumasta: Psion 5mx:n tapauksessa asennuspaketti eli `sis`-tiedosto voidaan joko ladata itse Psioniin ja käynnistää siellä tai vaihtoehtoisesti Psion voidaan kytkeä asennuspaketin sisältävään pöytäkoneeseen, jossa pyörii PsiWin-sovellus. Tällöin asennuspaketti voidaan käynnistää Window-

Kuva 11. Tiedoston luominen EPOCissa.

sista käsin. Riippumatta kummalla tavalla asennus käynnistetään, niin se etenee seuraavasti:

1. Käyttäjältä kysytään, mille asemalle sovellus asennetaan, ja samalla kerrotaan, paljonko sovellus vaatii tilaa. Mikäli laitteesta löytyy jo Buzz Bingon sama tai uudempi versio, paikallisen asennuksen tapauksessa EPOC varmistaa, että käyttäjä haluaa jatkaa asennusta. PsiWin sen sijaan ei kysele vaan asentaa orjallisesti. Sovelluksen versiotieto löytyy vastaavan `pkg`-tiedoston ensimmäiseltä riviltä heti sovelluksen nimen ja UID:n jälkeen. Buzz Bingon versionumero on 1.16.

2. Näytölle aukeaa dialogi, jossa on tiedoston `note.txt` (esimerkki 27, s. 92) sisältö. Tähän on syynä tiedoston `bingo.pkg` toinen rivi:

   ```
   "\vps\bingo\pkg\note.txt" - "", FT, TC
   ```

   Toisin sanoen `note.txt`:tä ei asenneta kohdekoneeseen vaan se näytetään dialogissa (FT), joka sisältää ainoastaan Continue-napin (TC). Asennukseen voidaan vaikuttaa monimutkaisemmillakin mekanismeilla, kuten antamalla käyttäjän valita, mitkä tiedostot asennetaan ja mitkä ei.

3. Sovelluksen osat siirretään kohdekoneeseen. Onnistuneen asennuksen jälkeen Buzz Bingo löytyy EPOCin ohjelmalistasta (Control panel → Add/remove) ja tietysti myös Extras-valikosta.

6 Lopuksi

EPOC-sovelluskehityksen suurin ongelma on tietynlainen epämääräisyys. Koska EPOC on laaja, sovellyskehystyyppinen järjestelmä, on sen ymmärtäminen ja oppiminen esim. perinteisiä kirjastoja hankalampaa. Hyvän EPOC-dokumentaation merkitystä ei voi liikaa korostaa. Ikävä kyllä, SDK:n mukana tuleva dokumentaatio ei läheskään kauttaaltaan ole tarpeeksi kattavaa tai eksaktia. Tämä on erityisen ikävää siitä syystä, että EPOC tuntuu kuitenkin pääosin sangen järkevästi rakennetulta käyttöjärjestelmältä, joka toki ansaitsisi kirjallistakin tukea. Voidaan toivoa, että tilanteeseen tulee parannusta ajan myötä sekä SDK:n dokumentaation kehittyessä että kirjatarjonnan parantuessa. Esimerkiksi hiljan ilmestynyt kirja *Professional Symbian Programming* [8] tuntuu sisältönsä puolesta lupaavalta mutta varmuus asiasta tulee luonnollisesti vasta, kun kirjan saa käsiinsä.

Osa EPOC-sovellusten rakentamiseen liittyvistä ongelmista voitaisiin luultavasti kiertää käyttämällä C++:n sijaan jotakin muuta kieltä eli lähinnä Javaa (ottamatta sen kummemmin kantaa OPL:n soveltuvuuteen). Javaa ja sulautettuja järjestelmiä, joihin EPOCkin tietyssä mielessä lukeutuu, on käsitelty yleisellä tasolla toisaalla [6]. Jo pelkästään osoitinaritmetiikan kirouksesta pääseminen eli villeiltä osoittimilta välttyminen on vahva argumentti Javan puolesta. Tosin asiaan perehtyminen voisi tuoda mukanaan myös ikäviä yllätyksiä esimerkiksi sen suhteen, kuinka hyvin Java-sovellukset sulautuisivat muiden EPOC-sovellusten joukkoon tiedostonkäsittelyn ja käyttöliittymän suhteen.

Vähemmän pessimistiseksi lopuksi voisi todeta, että alkuhankaluuksien jälkeen EPOC osaa tarjota miellyttäviäkin yllätyksiä: store-mekanismi ja tuki sovellusten paketoimiselle, esimerkiksi. Lisäksi täytyy muistaa, että tämä selvitys ainoastaan raapaisee EPOCin pintaa. Esimerkiksi rinnakkaisuuteen, tietokantoihin ja verkko-ohjelmointiin liittyvät asiat on sivuutettu kokonaan. Toisin sanoen EPOCissa riittää tutustumista ja pohdittavaa…

Lähteet

[1] Frank Buschmann, Regine Meunier, Hans Rohnert, Peter Sommerlad, and Michael Stal. *Pattern-Oriented Software Architecture — A System of Patterns*. John Wiley & Sons, 1996. 457 pages.

[2] IBM. 340 MB CF+ Type II one-inch hard drive. `http://www.storage.ibm.com/hardsoft/diskdrdl/micro/datasheet.htm`, June 1999.

[3] Glen E. Krasner and Stephen T. Pope. A cookbook for using the Model-View-Controller user interface paradigm in Smalltalk-80. *JOOP*, August/September 1988.

[4] Dominic Ostrowski, Maria Smith, and Neil Hepworth. EPOC developers course. Symbian, 1999.

[5] Psion. netBook Datasheet. `http://www.enterprise.psion.com/public/netbook/netbook.htm`, 1999.

[6] Vesa-Pekka Savikko. Java in embedded systems. Tampere University of Technology, 2000. Licentiate Thesis.

[7] Symbian. EPOC release 5 C++ SDK. `http://developer.epocworld.com/EPOClibrary/documentation/ER5/CPP/sysdoc/index.html`, 1999.

[8] Martin Tasker, Jonathan Allin, Jonathan Dixon, John Forrest, Mark Heath, Tim Richardson, and Mark Shackman. *Professional Symbian Programming*. Wrox Press, February 2000. 1060 pages.

Liitteet

A Painoindeksi

Luvussa 4.2 (alkaen sivulta 34) käsitellyn painoindeksisovelluksen lähdekoodi.

Esimerkki 10. Painoindeksisovelluksen resurssitiedosto.

```
// fats.rss

NAME FATS

#include <eikdef.rh>
#include <eikmenu.rh>
#include <eiktbar.rh>
#include <eikbutb.hrh>
#include <eikdialg.rh>
#include <eikctrls.hrh>
#include <eikedwin.rh>
#include <eikon.rsg>
#include <eikspace.rh>
#include <eikclock.rh>

#include "fats.hrh"

RESOURCE RSS_SIGNATURE {}

RESOURCE TBUF { buf=""; }

RESOURCE EIK_APP_INFO {
  hotkeys=fats_hotkeys;
  menubar=fats_menubar;
  toolbar=fats_toolbar;
}

RESOURCE HOTKEYS fats_hotkeys {
  control = {
    HOTKEY { command = EEikCmdExit; key='e'; }
  };
}

RESOURCE MENU_BAR fats_menubar {
  titles= {
    MENU_TITLE { menu_pane=fats_file_menu; txt="File"; }
  };
}

RESOURCE MENU_PANE fats_file_menu {
  items = {
    MENU_ITEM {
      command=EEikCmdExit;
      txt="Exit";
```

```
    }
  };
}

RESOURCE TOOLBAR fats_toolbar {
  controls = {
    TBAR_CTRL {
      id=EFatsFileName;
      type=EEikCtFileNameLabel;
      flags=EEikToolBarCtrlHasSetMinLength;
      length=KEikStdFileNameLabelHeight;
    },
    TBAR_BUTTON {
      flags=EEikToolBarCtrlHasSetMinLength;
      length=KEikStdToolBarButtonHeight;
      id=ECmdUpdate;
      txt="Update";
    },
    TBAR_CTRL {
      type=EEikCtSpacer;
      flags=EEikToolBarCtrlHasSetMinLength|EEikToolBarCtrlIsStretchable;
      length=0;
      control=SPACER;
    },
    TBAR_CTRL {
      type=EEikCtClock;
      control=CLOCK { digitalresourceid=R_EIK_DIGITAL_CLOCK;
                      analogresourceid=R_EIK_ANALOG_CLOCK; };
    },
    TBAR_CTRL {
      type=EEikCtSpacer;
      flags=EEikToolBarCtrlHasSetMinLength;
      length=KEikStdGapBelowClock;
      control=SPACER;
    }
  };
}

RESOURCE TBUF weight_label { buf = "Weight:"; }
RESOURCE TBUF height_label { buf = "Height:"; }
RESOURCE TBUF index_label  { buf = "Index:"; }
```

Esimerkki 11. Resurssitiedoston(kin) käyttämiä lukuvakioita.

```
// fats.hrh

#ifndef __FATS_HRH
#define __FATS_HRH

// Resurssitiedostossa käytetyt id:t
enum { ECmdUpdate=1, EFatsFileName };

#endif
```

Esimerkki 12. Luokkien esittelyt.

```
// fats.h

#ifndef __FATS_H
#define __FATS_H

#include <coecntrl.h>
#include <coeccntx.h>
#include <coemain.h>
#include <coecobs.h>

#include <eikappui.h>
#include <eikapp.h>
#include <eikdoc.h>
#include <eikcmds.hrh>
#include <eikchkbx.h>
#include <eiklabel.h>
#include <eikcapca.h>
#include <eikcapc.h>
#include <eikenv.h>
#include <eikctgrp.h>
#include <eikfpne.h>
#include <eikedwin.h>
#include <eikedwin.hrh>
#include <eikfnlab.h>
#include <eiktbar.h>
#include <eikcmbut.h>

#include  <fats.rsg>
#include "fats.hrh"

const TUid KUidFats = { 0x01000Ofa };

// class CFatsAppView

class CFatsAppView : public CCoeControl, public MCoeControlObserver,
                     public MCoeControlBrushContext {
public:
  ~CFatsAppView();
  void ConstructL( const TRect&  aRect );
  void UpdateIndex();
private:
  void HandleControlEventL(CCoeControl* aControl,TCoeEvent aEventType);
  TKeyResponse OfferKeyEventL(const TKeyEvent& aKeyEvent,TEventCode aType);
  CCoeControl* ComponentControl( TInt aIndex ) const;
  TInt CountComponentControls() const;
  void Draw(const TRect&  aRect ) const;

  enum TDirection { EUp = -1, EDown = 1 };
  TBool ValidateCurrent();
  void MoveFocus( TInt );
  void StepFocus( TDirection );
  CEikCapCArray* iLines;
  CEikFloatingPointEditor* iWeight;
```

```
54    CEikFloatingPointEditor* iHeight;
      CEikFloatingPointEditor* iIndex;
56    TDesC* iWeightLabel;
      TDesC* iHeightLabel;
58    TDesC* iIndexLabel;
      TInt iCurrentLine;
60  };

62  // class CFatsAppUi

64  class CFatsAppUi : public CEikAppUi {
    public:
66    void ConstructL();
      ~CFatsAppUi();
68  private:
      void HandleCommandL(TInt aCommand);
70
      CFatsAppView* iAppView;
72  };

74  // class CFatsDocument

76  class CFatsDocument : public CEikDocument {
    public:
78    CFatsDocument(CEikApplication& aApp);
    private:
80    CEikAppUi* CreateAppUiL();
    };
82
    // CFatsApplication
84
    class CFatsApplication : public CEikApplication {
86  private:
      CApaDocument* CreateDocumentL();
88    TUid AppDllUid() const;
    };
90
    #endif
92
```

Esimerkki 13. Lähdekoodi (pl. näkymä).

```
    // fats.cpp
2
    #include "fats.h"
4
    EXPORT_C CApaApplication* NewApplication() {
6     return new CFatsApplication;
    }
8
    GLDEF_C TInt E32Dll(TDllReason) {
10    return KErrNone;
    }
```

```
12
   // class CFatsApplication
14
   // AppDllUid
16 // Palauttaa sovelluksen UID:n
   TUid CFatsApplication::AppDllUid() const {
18   return KUidFats;
   }
20
   // CreateDocumentL
22 // Luo dokumentin. Vaikka FATS ei ole 'dokumenttipohjainen',
   // niin CFatsDocument-instanssi tarvitaan, koska sen vastuulla
24 // on seuraavan osan luominen jne.
   CApaDocument* CFatsApplication::CreateDocumentL() {
26   return new (ELeave) CFatsDocument(*this);
   }
28
   // class CFatsDocument
30
   // Rakentaja
32 CFatsDocument::CFatsDocument(CEikApplication& aApp)
     : CEikDocument(aApp) {}
34
   // CreateAppUiL
36 // Luo käyttöliittymän, joka luo varsinaiset näkymät
   CEikAppUi* CFatsDocument::CreateAppUiL() {
38   return new(ELeave) CFatsAppUi;
   }
40
   // class CFatsAppUi
42
   // ConstructL
44 // Kaksivaiheisen rakentamisen toinen osa
   void CFatsAppUi::ConstructL() {
46   // "Kantaluokan" rakentaja
     BaseConstructL();
48   // Luodaan näkymä
     iAppView=new(ELeave) CFatsAppView;
50   iAppView->ConstructL(ClientRect());
     // Sovelluksen nimi työkaluvalikkoon. Yleensä nimenä olisi
52   // käsiteltävän dokumentin/tiedoston nimi mutta tällä kertaa
     // tyydytään resurssitiedostossa annettuun nimeen
54   CEikFileNameLabel* filenameLabel =
       STATIC_CAST( CEikFileNameLabel*,
56                  iToolBar->ControlById( EFatsFileName ) );
     filenameLabel->UpdateL();
58   // Rekisteröidään näkymä kontrollipinoon, jotta se näkisi omat
     // näppäintapahtumansa (kts. CFatsAppView::OfferKeyEventL)
60   AddToStackL( iAppView );
   }
62
   // Purkaja
64 CFatsAppUi::~CFatsAppUi() {
     delete iAppView;
66 }
```

```
68 // HandleCommandL
   // Komentotapahtumat (esim. napit ja valikot) välittyvät näkymästä
70 // käyttöliittymälle. Kullakin tapahtumalla on oma yksilöllinen
   // tunnuslukunsa
72 void CFatsAppUi::HandleCommandL(TInt aCommand) {
     switch (aCommand) {
74     // EEikCmdExit on valmiiksi määritelty
     case EEikCmdExit:
76     Exit();
       break;
78     // ECmdUpdate on määritelty fats.hrh:ssa
     case ECmdUpdate:
80     iAppView->UpdateIndex();
       break;
82   }
   }
```

Esimerkki 14. Näkymän lähdekoodi.

```
   // fatsappview.cpp
2
   #include "fats.h"
4
   // class CFatsAppView
6
   // ConstructL
8  // Kaksivaiheisen rakentamisen toinen osa
   void CFatsAppView::ConstructL(const TRect& aRect) {
10   // Rakennetaan ikkuna
     CreateWindowL();
12   // Grafiikkakonteksti kuntoon: taustan ja kontrollien
     // piirtäminen ei tule muutoin 'siistiksi'
14   iContext = this;
     iBitmap = iEikonEnv->TexturedBitmap();
16   iBrushStyle = CGraphicsContext::EPatternedBrush;
     // Ikkuna täyttää koko ruudun
18   SetRectL(aRect);

20   // Varsinaisten kontrollien luonti
     // CEikFloatingPointEditor on numerokenttä,
22   // joka sisältää virhetarkastelut
     TReal value = 100;
24
     // Painokenttä
26   iWeight = new (ELeave) CEikFloatingPointEditor;
     iWeight->ConstructL( 0, 1000, 10 );
28   iWeight->SetValueL( &value );

30   // Pituuskenttä
     iHeight = new (ELeave) CEikFloatingPointEditor;
32   iHeight->ConstructL( 1, 1000, 10 );
     value = 184;
34   iHeight->SetValueL( &value );
```

```
36    // Indeksikenttä
      iIndex = new (ELeave) CEikFloatingPointEditor;
38    iIndex->ConstructL( 1,1000, 10 );

40    // CEikCaptionedControl:
      // Labelteksti   kontrolli trailer
42    // Esim:
      // Weight:  iWeight kg
44    CEikCaptionedControl* WLine = new (ELeave) CEikCaptionedControl;
      iWeightLabel = iCoeEnv->AllocReadResourceL( WEIGHT_LABEL );
46    WLine->SetCaptionL( *iWeightLabel );
      WLine->SetTrailerL( _L( "kg" ) );
48    WLine->iControl = iWeight;
      // Valinta hiirellä huomataan
50    WLine->SetObserver( this );

52    // Height: iHeight cm
      CEikCaptionedControl* HLine = new (ELeave) CEikCaptionedControl;
54    iHeightLabel = iCoeEnv->AllocReadResourceL( HEIGHT_LABEL );
      HLine->SetCaptionL( *iHeightLabel );
56    HLine->SetTrailerL( _L( "cm" ) );
      HLine->iControl = iHeight;
58    HLine->SetObserver( this );

60    // Index: iIndex
      CEikCaptionedControl* ILine = new (ELeave) CEikCaptionedControl;
62    iIndexLabel = iCoeEnv->AllocReadResourceL( INDEX_LABEL );
      ILine->SetCaptionL( *iIndexLabel );
64    ILine->iControl = iIndex;
      ILine->SetNonFocusing();
66
      // CEikCapCArray tallettaa CEikCaptionedControl:it taulukkoon
68    iLines = new (ELeave) CEikCapCArray( 3 );
      iLines->AppendL( WLine );
70    iLines->AppendL( HLine );
      iLines->AppendL( ILine );
72
      for( int i = 0; i < iLines->Count(); i++ ) {
74      (*iLines)[i]->iCaption->SetContainerWindowL( *this );
        (*iLines)[i]->iControl->SetContainerWindowL( *this );
76      (*iLines)[i]->SetContainerWindowL( *this );
      }
78
      // Taulukko vasempaan yläkulmaan minimikoossa
80    iLines->SetRectL( TRect( TPoint(), iLines->MinimumSize() ) );

82    // iCurrentLine kertoo valittuna olevan taulukkorivin indeksin
      iCurrentLine = 0;
84    // Merkitään vastaava rivi valituksi
      (*iLines)[iCurrentLine]->SetCurrent( ETrue );
86    // Kontrollin aktivointi
      ActivateL();
88  }

90  // Purkaja
```

```
// CEiCaptionedControl omistaa sisältämänsä kontrollit
// eli sen purkaja hoitaa niidenkin tuhoamiset
CFatsAppView::~CFatsAppView() {
  delete iLines;
  delete iWeightLabel;
  delete iHeightLabel;
  delete iIndexLabel;
}

// Draw
// Taustan piirto
void CFatsAppView::Draw(const TRect&  aRect ) const {
  iEikonEnv->FillTexturedRect( aRect );
}

// ComponentControl
// Palauttaa annettua indeksiä vastaavan kontrollin
CCoeControl* CFatsAppView::ComponentControl( TInt aIndex ) const {
  return (*iLines)[aIndex];
}

// CountComponentControl
// Palauttaa näkymän sisältämien kontrollien määrän
TInt CFatsAppView::CountComponentControls() const {
  return iLines->Count();
}

// ValidateCurrent
// Onko valittuna olevan tekstikentän (eli iLines-taulukon rivin
// iControl-jäsenmuuttuja) sisältö OK
TBool CFatsAppView::ValidateCurrent() {
  TRAPD( error, (*iLines)[iCurrentLine]->iControl->PrepareForFocusLossL() );
  if( error != KErrNone ) {
    return EFalse;
  }
  return ETrue;
}

// MoveFocus
// Valitaan annettu rivi, edellyttäen, että nykyinen rivi on OK
void CFatsAppView::MoveFocus( TInt aLine ) {
  if( !ValidateCurrent() ) {
    return;
  }
  CEikCaptionedControl* cap = (*iLines)[iCurrentLine];
  cap->SetCurrent( EFalse );
  iCurrentLine = aLine;
  (*iLines)[iCurrentLine]->SetCurrent( ETrue );
}

// HandleControlEventL
// Tapahtumienkäsittely
void CFatsAppView::HandleControlEventL( CCoeControl* aControl,
                                        TCoeEvent aEvent ) {
  switch( aEvent ) {
  case EEventRequestFocus:
```

```
      MoveFocus( iLines->FindLineIndex( aControl ) );
148     break;
    }
150 }

152 // OfferKeyEventL
    // Näppäintapahtumien käsittely
154 TKeyResponse CFatsAppView::OfferKeyEventL( const TKeyEvent& aKeyEvent,
                                               TEventCode aType ) {
156   switch( aKeyEvent.iCode ) {
    case EKeyTab:
158 case EKeyDownArrow:
      StepFocus( EDown );
160   return EKeyWasConsumed;
      break;
162 case EKeyUpArrow:
      StepFocus( EUp );
164   return EKeyWasConsumed;
      break;
166 case EKeyEnter:
      UpdateIndex();
168   return EKeyWasConsumed;
      break;
170 default:
      // Muut merkit välitetään eteenpäin, jotta kirjoittaminen onnistuisi
172   return (*iLines)[iCurrentLine]->iControl->OfferKeyEventL( aKeyEvent,
                                                               aType );
174 }
176 }

    // UpdateIndex
178 // Päivitetään painoindeksi
    void CFatsAppView::UpdateIndex() {
180   if( !ValidateCurrent() ) {
      return;
182   }
    TReal m = iHeight->Value() / 100;
184 TReal index = iWeight->Value() / ( m*m );
    iIndex->SetValueL( &index );
186 iIndex->DrawNow();
    }
188

    // StepFocus
190 // Valitaan seuraava/edellinen rivi
    // Toteutus pyrkii olemaan mahdollisimman
192 // yleiskäyttöinen: Rivien määrää ei olla rajattu
    // ja nonfocusing-kontrollit hypätään yli
194 void CFatsAppView::StepFocus( TDirection aDir ) {
    TInt iLine = iCurrentLine;
196 do {
      iLine += aDir;
198   if( iLine < 0 ) {
        iLine = iLines->Count() - 1;
200   }
      if( iLine >= iLines->Count() ) {
202     iLine = 0;
```

```
    }
  } while( (*iLines)[iLine]->IsNonFocusing() );
  MoveFocus( iLine );
}
```

B Buzz Bingo

Luvussa 5 (alkaen sivulta 41) käsitellyn Buzz Bingo -esimerkin lähdekoodit. Tiedostot on jaoteltu kolmeen ryhmään:

1. Malliin (*model, engine*) eli dynaamiseen kirjastoon `bingomodel.dll` liittyvät tiedostot.
2. Käyttöliittymä- ja MVC-mielessä myös kontrollerikoodi, joiden lopputuote on `bingo.app` EIKON-sovellus.
3. Asennus- ja konfigurointitiedostot.

Seuraavat tiedostot on listattu jo luvussa 5, joten niitä ei ole tarvis toistaa:

- `bingoaif.rss` (esim. 9, s. 53. Sovelluksen AIF-tiedoston (*Application Information File*) konfigurointititiedosto, joka kuvaa sovelluksen ominaisuudet.
- `bingo.pkg` (esim. 8, s. 52). Konfigurointitiedosto, jonka pohjalta `makesis` rakentaa EPOC-asennuspaketin (eli `sis`-tiedoston).

B.1 Kirjasto

Esimerkki 15. Dynaamisen kirjaston `mmp`-tiedosto.

```
TARGET          bingomodel.dll
TARGETTYPE      dll
UID             0x1000008d  0x10005fef
PROJECT         vps
SUBPROJECT      bingo
SOURCE          bingomodel.cpp
USERINCLUDE     .
SYSTEMINCLUDE   \epoc32\include
LIBRARY         euser.lib efsrv.lib estor.lib
```

Esimerkki 16. Kirjaston esittely.

```
// bingomodel.h

#ifndef _BINGOMODEL_H_
#define _BINGOMODEL_H_

#include <e32std.h>
#include <e32base.h>
#include <s32strm.h>

enum { KErrBingoDimension = 3941, KErrBingoWordfile };

#define DIMENSION_ASSERT( x, y ) \
          __ASSERT_DEBUG( (x) >= 0 && (x) < iDim && (y) >= 0 && (y) < iDim, \
                          User::Panic( _L( "Dimension assertion failed." ), \
                                       KErrBingoDimension ) )

// class CBingoModel
// Bingon tietorakenne tai malli.
// Pitää kirjaa pelitilanteesta ja tiedostonkäsittelystä
// eli peli voidaan tallettaa tarvittaessa.
// Sanatiedoston formaatti: termi per rivi
class CBingoModel : public CBase {
public:
  // Bingolapunluonti. dim saraketta ja riviä.
  // Sanatiedosto wordfile
  IMPORT_C void ConstructL( TInt dim, const TFileName wordfile );
  IMPORT_C ~CBingoModel();
  // Ruudun valinta/peruutus. Paluuarvo tosi, jos bingo.
  IMPORT_C TBool Toggle( TInt x, TInt y );
  // Onko jo tullut bingo?
  IMPORT_C TBool IsBingo() const;
  // Palauttaa annetussa ruudussa olevan termin
  IMPORT_C TDesC& Term( TInt x, TInt y );
  // Onko ruksittu?
  IMPORT_C TBool IsCrossed( TInt x, TInt y );
  // Palauttaa dimension
  IMPORT_C TInt Dimension() const;
  // Palauttaa sanatiedoston
  IMPORT_C TFileName Wordfile() const;
  // Puhdistetaan lappu eli poistetaan ruksit
  IMPORT_C void Clear();
  // Tuki sarjallistamiselle
  IMPORT_C void ExternalizeL( RWriteStream& out ) const;
  IMPORT_C void InternalizeL( RReadStream& in );
  IMPORT_C void SaveL( TDesC& filename );
  IMPORT_C TStreamId StoreL( CStreamStore& store) const;
  IMPORT_C void RestoreL(const CStreamStore& store, TStreamId id);
  IMPORT_C static CBingoModel* NewL( RReadStream& in );
  IMPORT_C static CBingoModel* NewL( TDesC& filename );

private:
  // class CCell.
  // Pelilapun ruutu: sana ja onko se ruksittu
```

```
54    class CCell : public CBase {
      public:
56      CCell();
        ~CCell();
58      void ExternalizeL( RWriteStream& out ) const;
        void InternalizeL( RReadStream& in );
60      TDesC* term;
        TBool crossed;
62    };
      // Sanaston lukeminen
64    void ReadWordsL( RFile& file, CArrayPtrFlat<HBufC>* words );
      // Pelilapun indeksointi
66    CCell& At( TInt x, TInt y ) {
        return *iGrid[x + y * iDim];
68    }

70    TFileName iWordfile;   // Sanatiedosto
      TInt iDim;             // Lapun dimensio
72    CCell** iGrid;         // Pelilappu
      TBool bBingo;          // Onko tullut bingo?
74
  };
76
  #endif
```

Esimerkki 17. Kirjaston toteutus.

```
  // CBingoModel.cpp
2
  #include <e32math.h>
4 #include <f32file.h>
  #include <s32stor.h>
6 #include <s32file.h>

8 #include "bingomodel.h"

10 EXPORT_C TInt E32Dll( TDllReason ) {
     return 0;
12 }

14 // ConstructL
   // Arpoo halutunkokoisen (dimension) pelilapun.
16 // Sanat/termit löytyvät tiedostosta wordfile (yksi/rivi)
   EXPORT_C void CBingoModel::ConstructL( TInt dimension,
18                                        const TFileName wordfile) {
     // Yhteys tiedostopalvelimeen
20   RFs fs;
     fs.Connect();
22   // Tiedoston avaaminen
     RFile file;
24   file.Open( fs, wordfile, EFileShareReadersOnly );
     // Luettujen sanojen lista
26   CArrayPtrFlat<HBufC> * words = new (ELeave)  CArrayPtrFlat<HBufC>( 10 );
```

```
    CleanupStack::PushL( words );
    // Luetaan sanat
    TRAPD( error, ReadWordsL( file, words ) );
    file.Close();
    fs.Close();
    if( error != KErrNone  || dimension * dimension > words->Count() ) {
      // Tuhotaan sanalistan sanat
      // Siivouspino hoitaa varsinaisen listan tuhoamisen
      words->ResetAndDestroy();
      User::Leave( KErrBingoWordfile );
    }
    iWordfile = wordfile;
    iGrid = new (ELeave)  CCell*[dimension * dimension];
    // iDim päivitetään vasta tässä sen takia, että jos
    // edellisen rivin muistinvaraus ei onnistu, niin
    // purkaja toimii silti, koska iDim == 0
    iDim = dimension;
    // Nollataan varmuuden vuoksi, jos muisti
    // ei riitä joka alkiolle
    for( TInt z = 0; z < iDim*iDim; z++ ) {
      iGrid[z] = 0;
    }
    // Uudet bingolapun ruudut
    for( TInt k = 0; k < iDim*iDim; k++ ) {
      iGrid[k] = new (ELeave) CCell;
    }
    TInt64 ticks = User::TickCount();
    for( TInt i = 0; i < iDim; i++ ) {
      for( TInt j = 0; j < iDim; j++ ) {
        TInt index = Math::Rand( ticks ) % words->Count();
        At( j, i ).term =  words->At( index );
        words->Delete( index );
      }
    }
    words->ResetAndDestroy();
    CleanupStack::PopAndDestroy();
  }

  // ReadWordsL
  // Luetaan annetun tiedoston kaikki rivit eli bingolapun
  // termit sanataulukkoon
  void CBingoModel::ReadWordsL( RFile& file, CArrayPtrFlat<HBufC>* words ) {
     // Lukeminen rivi kerrallaan. Rivin maksimipituus 256 merkkiä.
    TFileText ftxt;
    ftxt.Set( file );
    TBuf<256> buf;
    // Luetaan kaikki sanat listaan
    while( ftxt.Read( buf ) == KErrNone ) {
      HBufC* term = buf.AllocL();
      CleanupStack::PushL( term );
      words->AppendL( term );
      CleanupStack::Pop();  // term
    }
  }

  // Purkaja
```

```
EXPORT_C CBingoModel::~CBingoModel() {
  for( TInt i = 0; i < iDim * iDim; i++ ) {
    delete iGrid[i];
  }
  delete [] iGrid;
  iGrid = 0;
  iDim = 0;
}

// Toggle
// Rastii tai poistaa ruksin ruudusta. Palauttaa toden, jos syntyi bingo
EXPORT_C TBool CBingoModel::Toggle( TInt x, TInt y ) {
  DIMENSION_ASSERT( x, y );
  if( At( x, y ).crossed ) {
    At( x, y ).crossed = EFalse;
    return EFalse;
  }
  At( x, y ).crossed = ETrue;
  // Tarkistetaan bingot
  TBool xbingo = ETrue;
  TBool ybingo = ETrue;
  TBool dbingo1 = ETrue;
  TBool dbingo2 = ETrue;
  for( TInt i = 0; i < iDim; i++ ) {
    // Vaaka
    if( !At( i, y ).crossed ) {
      xbingo = EFalse;
    }
    // Pysty
    if( !At( x, i ).crossed ) {
      ybingo = EFalse;
    }
    // Ylävasemmalta alaoikealle
    if( !At( i, i ).crossed ) {
      dbingo1 = EFalse;
    }
    // Yläoikealta alavasemmalle
    if( !At( iDim - i - 1, i ).crossed ) {
      dbingo2 = EFalse;
    }
  }
  // Bingo ei poistu
  if( !bBingo ) {
    bBingo = xbingo || ybingo || dbingo1 || dbingo2;
  }
  return bBingo;
}

// IsBingo
// Onko bingo?
EXPORT_C TBool CBingoModel::IsBingo() const {
  return bBingo;
}

// Term
// Palauttaa annetussa ruudussa olevan termin
```

```
EXPORT_C TDesC& CBingoModel::Term( TInt x, TInt y ) {
  DIMENSION_ASSERT( x, y );
  return *At( x, y ).term;
}

// IsCrossed
// Onko ruutu ruksittu?
EXPORT_C TBool CBingoModel::IsCrossed( TInt x, TInt y ) {
  DIMENSION_ASSERT( x, y );
  return At( x, y ).crossed;
}

// Dimension
// Palauttaa lapun dimension
EXPORT_C TInt CBingoModel::Dimension() const {
  return iDim;
}

// Wordfile
// Palauttaa sanatiedoston
EXPORT_C TFileName CBingoModel::Wordfile() const {
  return iWordfile;
}

// Clear
// Puhdistaa pelilapun eli poistaa ruksit
EXPORT_C void CBingoModel::Clear() {
  for( TInt x = 0; x < iDim; x++ ) {
    for( TInt y = 0; y < iDim; y++ ) {
      At( x, y ).crossed = EFalse;
    }
  }
  bBingo = EFalse;
}

// ExternalizeL
// Olion kirjoittaminen streamiin
EXPORT_C void CBingoModel::ExternalizeL( RWriteStream& out ) const {
  out.WriteInt32L( iDim );
  out << iWordfile;
  for( TInt i = 0; i < iDim * iDim; i++ ) {
    out << *iGrid[i];
  }
}

// InternalizeL
// Olion lukeminen streamista
EXPORT_C void CBingoModel::InternalizeL( RReadStream& in ) {
  iDim = in.ReadInt32L();
  in >> iWordfile;
  iGrid = new (ELeave)  CCell*[iDim*iDim];
  for( TInt j = 0; j < iDim*iDim; j++ ) {
    iGrid[j] = new (ELeave) CCell;
  }
  for( TInt i = 0; i < iDim * iDim; i++ ) {
    in >> *iGrid[i];
```

```
    }
}

// CBingoModel::NewL
// Luo uuden bingolapun annetusta streamistä
EXPORT_C CBingoModel* CBingoModel::NewL( RReadStream& in ) {
  CBingoModel* bingo = new (ELeave) CBingoModel;
  bingo->InternalizeL( in );
  return bingo;
}

// CBingoModel::NewL
// Luo uuden bingolapun annetusta tiedostosta
EXPORT_C CBingoModel* CBingoModel::NewL( TDesC& filename ) {
  RFs fs;
  fs.Connect();
  CFileStore* store = CDirectFileStore::OpenLC( fs, filename,
                                                EFileShareReadersOnly );
  RStoreReadStream in;
  in.OpenLC( *store, store->Root() );
  CBingoModel* bingo = NewL( in );
  CleanupStack::PopAndDestroy();
  CleanupStack::PopAndDestroy();
  fs.Close();
  return bingo;
}

// SaveL
// Tallettaa bingolapun annettuun tiedostoon
EXPORT_C void CBingoModel::SaveL( TDesC& filename ) {
  RFs fs;
  fs.Connect();
  CFileStore* store = CDirectFileStore::ReplaceLC( fs,
                                                   filename,
                                                   EFileShareAny | EFileWrite );
  store->SetTypeL(KDirectFileStoreLayoutUid);
  RStoreWriteStream out;
  TStreamId id = out.CreateLC( *store );
  ExternalizeL( out );
  out.CommitL();
  store->SetRootL( id );
  store->CommitL();
  CleanupStack::PopAndDestroy(); // id
  CleanupStack::PopAndDestroy(); // store
  fs.Close();
}

// StoreL
// Olion kirjoittaminen storeen
EXPORT_C TStreamId CBingoModel::StoreL( CStreamStore& store ) const {
  RStoreWriteStream out;
  TStreamId id = out.CreateLC( store );
  ExternalizeL( out );
  out.CommitL();
  CleanupStack::PopAndDestroy();
  return id;
```

```
}

// RestoreL
// Olion lukeminen storesta
EXPORT_C void CBingoModel::RestoreL( const CStreamStore& store,
                                     TStreamId id) {
  RStoreReadStream in;
  in.OpenLC( store, id );
  InternalizeL( in );
  CleanupStack::PopAndDestroy();
}

// class BingoModel::CCell
// Pelilapun ruutu

// Rakentaja
CBingoModel::CCell::CCell() : term( 0 ), crossed( EFalse ) {}

// Purkaja
CBingoModel::CCell::~CCell() { delete term; term = 0; }

// ExternalizeL
// Olion kirjoittaminen streamiin
void CBingoModel::CCell::ExternalizeL( RWriteStream& out ) const {
  out << *term;
  out.WriteInt8L( crossed );
}

// InternalizeL
// Olion lukeminen streamista
void CBingoModel::CCell::InternalizeL( RReadStream& in ) {
  term = HBufC::NewL( in, KMaxTInt );
  crossed = (TBool)in.ReadInt8L();
}
```

B.2 Käyttöliittymä

Esimerkki 18. Sovelluksen `mmp`-tiedosto.

```
TARGET          bingo.app
TARGETTYPE      app
UID             0x1000006c 0x10005ff0
TARGETPATH      \system\apps\bingo
PROJECT         vps
SUBPROJECT      bingo
SOURCE          bingo.cpp bingogame.cpp bingoappui.cpp
RESOURCE        bingo.rss
USERINCLUDE     .
SYSTEMINCLUDE   \epoc32\include
LIBRARY         euser.lib apparc.lib cone.lib eikon.lib bingomodel.lib
LIBRARY         estor.lib efsrv.lib bafl.lib
```

Esimerkki 19. Resurssitiedosto.

```
// bingo.rss
2
NAME BING
4
#include <eikdef.rh>
6 #include <eikmenu.rh>
#include <eiktbar.rh>
8 #include <eikbutb.hrh>
#include <eikdialg.rh>
10 #include <eikctrls.hrh>
#include <eikedwin.rh>
12 #include <eikon.rsg>
#include <eikspace.rh>
14 #include <eikclock.rh>
#include <eikmfne.rh>
16 #include <eikfsel.rh>
#include <eiklabel.rh>
18
#include "bingo.hrh"
20
RESOURCE RSS_SIGNATURE {}
22
RESOURCE TBUF { buf = ""; }
24
RESOURCE EIK_APP_INFO
26   {
  hotkeys=bingo_hotkeys;
28   menubar=bingo_menubar;
  toolbar=bingo_toolbar;
30   }

32 RESOURCE HOTKEYS bingo_hotkeys {
  control = {
34     HOTKEY { command = EEikCmdFileNew; key='n'; },
    HOTKEY { command = EEikCmdFileOpen; key='o'; },
36     HOTKEY { command = EEikCmdFileSave; key='s'; },
    HOTKEY { command = EEikCmdFileSaveAs; key='a'; },
38     HOTKEY { command = EEikCmdExit; key='e'; }
  };
40 }

42 RESOURCE MENU_BAR bingo_menubar {
  titles= {
44     MENU_TITLE { menu_pane=bingo_menu; txt="Bingo"; }
  };
46 }

48 RESOURCE MENU_PANE bingo_menu {
  items = {
50     MENU_ITEM {
      command=EEikCmdFileNew;
52       txt="New...";
    },
```

```
      MENU_ITEM {
        command=EEikCmdFileOpen;
        txt="Open...";
      },
      MENU_ITEM {
        command=EEikCmdFileSave;
        txt="Save";
      },
      MENU_ITEM {
        command=EEikCmdFileSaveAs;
        txt="Save As...";
        flags = EEikMenuItemSeparatorAfter;
      },
      MENU_ITEM {
        command=EBingoAbout;
        txt="About...";
        flags = EEikMenuItemSeparatorAfter;
      },
      MENU_ITEM {
        command=EEikCmdExit;
        txt="Exit";
      }
    };
  }

  RESOURCE TOOLBAR bingo_toolbar {
    controls = {
      TBAR_CTRL {
        id=EBingoFileName;
        type=EEikCtFileNameLabel;
        flags=EEikToolBarCtrlHasSetMinLength;
        length=KEikStdFileNameLabelHeight;
        },
      TBAR_BUTTON {
        flags=EEikToolBarCtrlHasSetMinLength;
        length=KEikStdToolBarButtonHeight;
        id=EBingoNewWords;
        txt="New\nWords";
      },
      TBAR_BUTTON {
        flags=EEikToolBarCtrlHasSetMinLength;
        length=KEikStdToolBarButtonHeight;
        id=EBingoReplay;
        txt="Replay";
      },

      TBAR_CTRL {
        type=EEikCtSpacer;
        flags=EEikToolBarCtrlHasSetMinLength|EEikToolBarCtrlIsStretchable;
        length=0;
        control=SPACER;
        },
      TBAR_CTRL {
        type=EEikCtClock;
        control=CLOCK { digitalresourceid=R_EIK_DIGITAL_CLOCK;
                        analogresourceid=R_EIK_ANALOG_CLOCK; };
```

```
110         },
        TBAR_CTRL {
112         type=EEikCtSpacer;
            flags=EEikToolBarCtrlHasSetMinLength;
114         length=KEikStdGapBelowClock;
            control=SPACER;
116       }
      };
118   }

120   RESOURCE DIALOG r_new_bingo_dialog {
        title = "New Bingo";
122     buttons = R_EIK_BUTTONS_BROWSE_CANCEL_OK;
        flags = EEikDialogFlagWait;
124     items= {
          DLG_LINE {
126         type=EEikCtNumberEditor;
            prompt="Dimension";
128         id=EBingoDimensionEditor;
            control=NUMBER_EDITOR {
130           min=1;
              max=10;
132         };
            itemflags = EEikDlgItemSeparatorAfter;
134       },
          DLG_LINE {
136         type = EEikCtLabel;
            prompt = "Word file";
138         control = LABEL {};
          },
140       DLG_LINE {
            type =  EEikCtFileNameSel;
142         prompt = "Name";
            control = FILENAMESELECTOR {};
144         id = EBingoFileNameSelector;
          },
146       DLG_LINE {
            type=EEikCtFolderNameSel;
148         prompt="Folder";
            id=EBingoFolderNameSelector;
150         control=FOLDERNAMESELECTOR {};
          },
152       DLG_LINE {
            type=EEikCtDriveNameSel;
154         prompt="Disk";
            id=EBingoDriveNameSelector;
156         control=DRIVENAMESELECTOR {};
          }
158     };
      }
160

      RESOURCE TBUF r_bingo_unchanged { buf = "No changes need to be saved."; }
162   RESOURCE TBUF r_bingo_bingo_title { buf = "Congratulations!"; }
      RESOURCE TBUF r_bingo_bingo_message { buf = "BINGO!"; }
164   RESOURCE TBUF r_bingo_error_title { buf = "Error"; }
      RESOURCE TBUF r_bingo_open_error_prefix { buf = "Can't open "; }
```

```
RESOURCE TBUF r_bingo_wordfile_error { buf = "Problems with wordfile.\n"
                                       "\nPossible reasons:\n\n"
                                       "  - Out of memory (wordfile too big) -\n"
                                       "  - Not enough words -\n"
                                       "  - Not a text file -"; }
RESOURCE TBUF r_bingo_about_title { buf = "About..."; }
RESOURCE TBUF r_bingo_about_message { buf = "Buzz Bingo\n\n"
                                      "by Vespe Savikko (vespe@iki.fi)\n"
                                      "\nwith\n\n"
                                      "Matti Rintala (bitti@cs.tut.fi)\n"
                                      "as \"The Voice\""; }
```

Esimerkki 20. Resurssitiedoston ja muun lähdekoodin jakamia vakioita.

```
// bingo.hrh

enum { EBingoDimensionEditor = 1000,
       EBingoFileNameSelector,
       EBingoFolderNameSelector,
       EBingoDriveNameSelector,
       EBingoFileName,
       EBingoReplay,
       EBingoNewWords,
       EBingoAbout };
```

Esimerkki 21. Dokumentti- ja sovellusluokkien esittelyt.

```
// bingo.h

#ifndef __BINGO_H
#define __BINGO_H

#include <coecntrl.h>
#include <coeccntx.h>
#include <coemain.h>
#include <coecobs.h>

#include <eikappui.h>
#include <eikapp.h>
#include <eikdoc.h>
#include <eikcmds.hrh>
#include <eikenv.h>
#include <eikctgrp.h>
#include <eikfnlab.h>
#include <eikfsel.h>
#include <eikfbrow.h>
#include <eiktbar.h>
#include <eikcmbut.h>
#include <eikctrls.hrh>
#include <eikedwin.h>
```

```
#include <eikedwin.hrh>
#include <eikdialg.hrh>
#include <eikdialg.h>
#include <eikon.rsg>
#include <eikcfdlg.h>
#include <eikinfo.h>
#include <eikproc.h>
#include <eiksndpy.h>
#include <eikappui.h>
#include <eikapp.h>

#include <baspyrc.h>
#include <d32snd.h>
#include <s32file.h>

#include <apparc.h>

#include  <bingo.rsg>
#include "bingo.hrh"
#include "bingomodel.h"

const TUid KUidBingo = { 0x10005ff0 };

#define KDefaultDimension  4
#define KDefaultWordfile   "words.txt"
#define KDefaultSoundfile  "bingo.snd"

//
// CBingoDocument
//

class CBingoDocument : public CEikDocument  {
public:
  CBingoDocument(CEikApplication& aApp);
  ~CBingoDocument();
  void ConstructL();
  CBingoModel* Model() { return iModel; }
  void SetModelL( TInt aDimension, const TFileName& aWordfile );
  void SetModelL( const TFileName& aFilename );
  void SaveModelL( const TFileName& aFilename );
  TBool IsCurrentFile( const TFileName& aFilename ) const;
  TFileName Directory() const;
  void NewFileL( const TFileName& aFilename );
private:
  void StoreL( CStreamStore& aStore,
               CStreamDictionary& aDic ) const;
  void RestoreL( const CStreamStore& aStore,
                 const CStreamDictionary& aDic );
  CEikAppUi* CreateAppUiL();

  CBingoModel* iModel;

};

//
// CBingoApplication
```

```
80 //

82 class CBingoApplication : public CEikApplication  {
   private:
84   CApaDocument* CreateDocumentL();
     TUid AppDllUid() const;
86 };

88 #endif
```

Esimerkki 22. Dokumentti- ja sovellusluokkien toteutukset.

```
   // bingo.cpp
2
   #include "bingo.h"
4  #include "bingoappui.h"
   #include "bingogame.h"
6
   EXPORT_C CApaApplication* NewApplication() {
8    return new CBingoApplication;
   }
10
   GLDEF_C TInt E32Dll(TDllReason) {
12   return KErrNone;
   }
14
   // class CBingoApplication
16
   // AppDllUid
18 // Palauttaa sovelluksen UID:n
   TUid CBingoApplication::AppDllUid() const {
20   return KUidBingo;
   }
22
   // CreateDocumentL
24 // Luo dokumentin.
   CApaDocument* CBingoApplication::CreateDocumentL() {
26   CBingoDocument* doc = new (ELeave) CBingoDocument( *this );
     doc->ConstructL();
28   return doc;
   }
30
   // class CBingoDocument
32
   // Rakentaja
34 CBingoDocument::CBingoDocument(CEikApplication& aApp)
     : CEikDocument( aApp ) {}
36
   // ConstructL
38 // Toinen vaihe
   void CBingoDocument::ConstructL() {
40   if( !iModel ) {
       TFileName appname = Application()->AppFullName();
```

```
42      TParse parse;
        parse.Set( appname, NULL, NULL );
44      TFileName wordfile = parse.DriveAndPath();
        wordfile += _L( KDefaultWordfile );
46      SetModelL( KDefaultDimension, wordfile );
      } else {
48      SetModelL( iModel->Dimension(), iModel->Wordfile() );
      }
50  }

52  // Purkaja
    CBingoDocument::~CBingoDocument() {
54    delete iModel;
      iModel = 0;
56  }

58  // NewFileL
    // Luo uuden mallin ja annetun nimisen tallennustiedoston
60  void CBingoDocument::NewFileL( const TFileName& aFilename ) {
      // Uusi store
62    CFileStore* store =  CreateFileStoreLC( Process()->FsSession(),
                                              aFilename );
64    CleanupStack::Pop();
      // Tuhotaan vanha päästore ja korvataan
66    // se uudella
      delete(((CEikProcess*)Process())->MainStore());
68    ((CEikProcess*)Process())->SetMainStore( store );
      Process()->SetMainDocFileName( aFilename );
70  }

72  // StoreL
    // Tallennus streamiin eli sarjallistaminen
74  void CBingoDocument::StoreL( CStreamStore& aStore,
                                 CStreamDictionary& aDic ) const {
76    TStreamId id = iModel->StoreL( aStore );
      aDic.AssignL( KUidBingo, id );
78  }

80  // RestoreL
    // Luku streamistä
82  void CBingoDocument::RestoreL( const CStreamStore& aStore,
                                   const CStreamDictionary& aDic ) {
84
      CBingoModel* model = new (ELeave) CBingoModel;
86    CleanupStack::PushL( model );
      TStreamId id = aDic.At( KUidBingo );
88    model->RestoreL( aStore, id );
      CleanupStack::Pop();
90    delete iModel;
      iModel = model;
92    SetChanged( EFalse );
    }
94
    // SetModelL
96  // Uusi malli
    void CBingoDocument::SetModelL( TInt aDimension,
```

```
98                                  const TFileName& aWordfile ) {
    CBingoModel* model = new (ELeave) CBingoModel;
100   CleanupStack::PushL( model );
    model->ConstructL( aDimension, aWordfile );
102   CleanupStack::Pop();
    delete iModel;
104   iModel = model;
  }
106
  // SetModelL
108 // Uusi malli, joka luetaan annetusta tiedostosta
  void CBingoDocument::SetModelL( const TFileName& aFilename ) {
110   CFileStore* store = NULL;
    CStreamDictionary* dic =
112     CApaProcess::ReadRootStreamLC( Process()->FsSession(),
                                       store,
114                                    aFilename,
                                       EFileShareExclusive | EFileRead );
116   CleanupStack::PushL( store );
    RestoreL( *store, *dic );
118   CleanupStack::Pop();  // store
    CleanupStack::PopAndDestroy();  // dic
120   delete(((CEikProcess*)Process())->MainStore());
    ((CEikProcess*)Process())->SetMainStore( store );
122   Process()->SetMainDocFileName( aFilename );
  }
124
  // IsCurrentFile
126 // Onko annettu tiedoston nimi dokumentin nimi?
  TBool CBingoDocument::IsCurrentFile( const TFileName& aFileName ) const {
128   return aFileName == Process()->MainDocFileName();
  }
130
  // Directory
132 // Palauttaa dokumentin hakemiston
  TFileName CBingoDocument::Directory() const {
134   TFileName filename = Process()->MainDocFileName();
    TParse parse;
136   parse.Set( filename, NULL, NULL );
    filename = parse.DriveAndPath();
138   return filename;
  }
140
  // SaveModelL
142 // Mallin tallennus
  void CBingoDocument::SaveModelL( const TFileName& aFilename ) {
144   ((CEikProcess*)Process())->SaveToDirectFileStoreL( this, &aFilename );
    SetChanged( EFalse );
146 }

148 // CreateAppUiL
  // Luo käyttöliittymän, joka luo varsinaiset näkymät
150 CEikAppUi* CBingoDocument::CreateAppUiL() {
      return new(ELeave) CBingoAppUi;
152 }
```

Esimerkki 23. Pääkäyttöliittymän esittely.

```
// bingoappui.h

#ifndef __BINGOAPPUI_H
#define __BINGOAPPUI_H

#include "bingo.h"
#include "bingogame.h"

//
// class CBingoAppUi
//

class CBingoAppUi : public CEikAppUi, MCoeControlObserver,
                    MSoundPlayerObserver {
public:
  void ConstructL();
  ~CBingoAppUi();

private:
  TBool ProcessCommandParametersL( TApaCommand aCommand,
                                   TFileName& aDocumentName,
                                   const TDesC& aTail );
  void HandleControlEventL( CCoeControl* aControl,
                            TCoeEvent aEventType );
  void HandleCommandL( TInt aCommand );
  void HandleModelChangeL();
  void CreateFileL( const TDesC& aFilename );
  void OpenFileL( const TDesC& aFilename );
  void OpenFileErrorL( const TFileName& aFilename ) const ;
  void NewWordsL();
  void NewBingoL();
  void SaveBingoL();
  void SaveAsBingoL();
  void OpenBingoL();
  void ErrorL( TDesC& aMsg ) const;
  void BingoL();
  void ReplayL();
  void PlaySoundL();
  void PlayComplete( TInt aStatus );
  void AboutL();

  // class CNewBingoDialog
  // Uuden lapun luomiseen tarvittava dialogi,
  // jonka avulla käyttäjä voi valita dimension
  // ja sanatiedoston
  class CNewBingoDialog : public CEikDialog {
    public:
      CNewBingoDialog( TFileName* aFilename, TInt* aDim );
      void PreLayoutDynInitL();
      TBool OkToExitL( TInt );

    private:
      TFileName* iFilename;
```

```
54        TInt* iDim;
          CEikDriveNameSelector* iDriveS;
56        CEikFolderNameSelector* iFolderS;
          CEikFileNameSelector* iFileS;
58      };

60    CBingoGame* iBingo;
      CBingoModel* iModel;
62    TDesC* iInitialPath;
      TFileName iSoundfile;
64    CSoundPlayer* iSoundPlayer;
  };
66
  #endif
```

Esimerkki 24. Pääkäyttöliittymän toteutus.

```
  // bingoappui.cpp
2
  #include "bingo.h"
4 #include "bingoappui.h"

6 // ConstructL
  // Kaksivaiheisen rakentamisen toinen osa
8 void CBingoAppUi::ConstructL() {
    // Kantaluokan toinen vaihe
10  BaseConstructL();
    // Luodaan näkymä
12  iBingo= new (ELeave) CBingoGame;
    iBingo->ConstructL( ClientRect() );
14  iBingo->SetObserver( this );
    // Dokumentin nimi toolbariin. Aluksi oletusdokumentti
16  CEikFileNameLabel* filenameLabel =
      STATIC_CAST( CEikFileNameLabel*,
18                 iToolBar->ControlById( EBingoFileName ) );
    filenameLabel->UpdateL();
20  // Äänitiedosto
    TFileName appname = iDocument->Application()->AppFullName();
22  TParse parse;
    parse.Set( appname, NULL, NULL );
24  iSoundfile = parse.DriveAndPath();
    iSoundfile += _L( KDefaultSoundfile );
26 }

28 // Purkaja
  CBingoAppUi::~CBingoAppUi() {
30  if( iDoorObserver ) {
      iDoorObserver->NotifyExit(MApaEmbeddedDocObserver::EKeepChanges);
32  }
    delete iBingo;
34 }

36 // ErrorL
```

```
// Virheikkuna
void CBingoAppUi::ErrorL( TDesC& aMsg ) const {
  TDesC* title = iEikonEnv->AllocReadResourceLC( R_BINGO_ERROR_TITLE );
  CEikDialog* dialog = new (ELeave) CEikInfoDialog( *title, aMsg);
  dialog->ExecuteLD( R_EIK_DIALOG_INFO );
  CleanupStack::PopAndDestroy();
}

// BingoL
// Onnitteluikkuna: Bingo!
void CBingoAppUi::BingoL() {
  // Soitetaan ääni
  RDevSound rs;
  rs.Open();
  TSoundConfig config;
  rs.Config( config );
  config().iVolume = EVolumeLoud;
  config().iAlawBufferSize = 8000;
  rs.SetConfig( config );
  iSoundPlayer = CSoundPlayer::NewL( *this, rs );
  CleanupStack::PushL( iSoundPlayer  );
  PlaySoundL();
  // dialogi
  TDesC* title = iEikonEnv->AllocReadResourceLC( R_BINGO_BINGO_TITLE );
  TDesC* msg = iEikonEnv->AllocReadResourceLC( R_BINGO_BINGO_MESSAGE );
  CEikDialog* dialog = new (ELeave) CEikInfoDialog( *title, *msg );
  dialog->ExecuteLD( R_EIK_DIALOG_INFO );
  iSoundPlayer->Stop();
  CleanupStack::PopAndDestroy(); // iSoundPlayer;
  iSoundPlayer = 0;
  rs.Close();
  CleanupStack::PopAndDestroy(); // msg
  CleanupStack::PopAndDestroy(); // title
}

// PlaySoundL
// Soittaa äänitiedoston iSoundfile
void CBingoAppUi::PlaySoundL() {
  // Tarkastetaan varmuuden vuoksi, että
  // iSoundPlayer on olemassa
  if( iSoundPlayer ) {
    iSoundPlayer->PlayFileL( iSoundfile,
                             iDocument->Process()->FsSession() );
  }
}

// PlayComplete
// MSoundPlayerObserveristä peritty.
// Kutsutaan, kun äänitiedosto on soitettu.
void CBingoAppUi::PlayComplete( TInt aStatus ) {
  if( aStatus == KErrNone ) {
    PlaySoundL();
  }
}

// AboutL
```

```
// About-ikkuna. Yksinkertainen viesti-ikkuna
void CBingoAppUi::AboutL() {
  TDesC* title = iEikonEnv->AllocReadResourceLC( R_BINGO_ABOUT_TITLE );
  TDesC* msg = iEikonEnv->AllocReadResourceLC( R_BINGO_ABOUT_MESSAGE );
  CEikDialog* dialog = new (ELeave) CEikInfoDialog( *title, *msg );
  dialog->ExecuteLD( R_EIK_DIALOG_INFO );
  CleanupStack::PopAndDestroy(); // msg
  CleanupStack::PopAndDestroy(); // title
}

// HandleControlEventL
// Näkymästä tulleet tapahtumat
void CBingoAppUi::HandleControlEventL( CCoeControl*,
                                       TCoeEvent event ) {
  if( event == EEventStateChanged ) {
    // Näkymä on muuttunut -> dokumenttikin on muuttunut
    SetDocChanged();
    // Tuliko bingo?
    if( iModel->IsBingo() ) {
      BingoL();
    }
  }
}

// HandleCommandL
// Komentotapahtumat (esim. napit ja valikot) välittyvät näkymästä
// käyttöliittymälle. Kullakin tapahtumalla on oma yksilöllinen
// tunnuslukunsa
void CBingoAppUi::HandleCommandL(TInt aCommand) {
  switch (aCommand) {
  case EEikCmdExit:
    SaveL();  // CEikAppUi
    Exit();   // CEikAppUi
    break;
  case EEikCmdFileNew:
    NewBingoL();
    break;
  case EBingoNewWords:
    NewWordsL();
    break;
  case EBingoReplay:
    ReplayL();
    break;
  case EEikCmdFileOpen:
    OpenBingoL();
    break;
  case EEikCmdFileSave:
    SaveBingoL();
    break;
  case EEikCmdFileSaveAs:
    SaveAsBingoL();
    break;
  case EBingoAbout:
    AboutL();
    break;
  }
```

```
}

TBool CBingoAppUi::ProcessCommandParametersL( TApaCommand aCommand,
                                              TFileName& aDocumentName,
                                              const TDesC& ) {
  return CEikAppUi::ProcessCommandParametersL(aCommand,aDocumentName);
}

// HandleModelChangeL
// Malli on muuttunut, joten käyttöliittymä ja näkymä
// täytyy päivittää vastaavasti
void CBingoAppUi::HandleModelChangeL() {
  iModel = STATIC_CAST( CBingoDocument*, iDocument)->Model();
  iBingo->SetModel( iModel );
  iBingo->DrawNow();
  iEikonEnv->UpdateTaskNameL();
  CEikFileNameLabel* filenameLabel =
    STATIC_CAST( CEikFileNameLabel*,
                 iToolBar->ControlById( EBingoFileName ) );
  filenameLabel->UpdateL();
  filenameLabel->DrawNow();
}

// CreateFileL
// Luodaan uusi dokumentti. Tätä kutsutaan bingon
// tapauksessa ainoastaan silloin, kun EPOC haluaa
// luodaa uuden bingotyyppisen tiedoston
void CBingoAppUi::CreateFileL( const TDesC& aFilename ) {
  SaveAnyChangesL();
  STATIC_CAST( CBingoDocument*, iDocument )->NewFileL( aFilename );
  HandleModelChangeL();
}

// OpenBingoL
// Ladataan Bingo
void CBingoAppUi::OpenBingoL() {
  TFileName filename =
    STATIC_CAST( CBingoDocument*, iDocument )->Directory();
  CEikFileOpenDialog* dialog =
    new (ELeave) CEikFileOpenDialog( &filename,
                                     R_EIK_TBUF_OPENING_FILE );
  dialog->RestrictToNativeDocumentFiles();
  if( dialog->ExecuteLD( R_EIK_DIALOG_FILE_OPEN ) ) {
    TRAPD( error, OpenFileL( filename ) );
    if( error != KErrNone ) {
      OpenFileErrorL( filename );
    }
  }
}

// OpenFileErrorL
// Annetun tiedoston avaamisen epäonnistumisesta kertova dialogi
void CBingoAppUi::OpenFileErrorL( const TFileName& aFilename ) const {
  TBuf<356> msg =
    *iEikonEnv->AllocReadResourceLC( R_BINGO_OPEN_ERROR_PREFIX );
  msg += aFilename;
```

```
    ErrorL( msg );
    CleanupStack::PopAndDestroy();
  }

  // OpenFileL
  // Luetaan Bingo tiedostosta. Tätä funktiota kutsuu
  // OpenBingoL (kts. yllä)
  void CBingoAppUi::OpenFileL( const TDesC& aFilename ) {
    // Talletetaan muutokset
    SaveAnyChangesL(); // CEikAppUi
    // Alustetaan dokumentti annetun tiedoston pohjalta
    STATIC_CAST( CBingoDocument*, iDocument )->SetModelL( aFilename );
    // Päivitetään näkymä
    HandleModelChangeL();
  }

  // NewWordsL
  // Uusi arvonta. Sanasto ja dimensiot säilyvät samoina
  void CBingoAppUi::NewWordsL() {
    STATIC_CAST( CBingoDocument*, iDocument )->ConstructL();
    HandleModelChangeL();
  }

  // ReplayL
  // Uusi peli samalla lapulla
  void CBingoAppUi::ReplayL() {
    iModel->Clear();
    HandleModelChangeL();
  }

  // SaveBingoL
  // Tallennus
  void CBingoAppUi::SaveBingoL() {
    if ( Document()->HasChanged() ) {
      SaveL();  // CEikAppUi
      iEikonEnv->InfoMsg( R_EIK_TBUF_FILE_SAVED );
    } else {
      iEikonEnv->InfoMsg( R_BINGO_UNCHANGED );
    }
  }

  // SaveAsBingoL
  // Tallennnus nimellä
  void CBingoAppUi::SaveAsBingoL() {
    TFileName filename =
      STATIC_CAST( CBingoDocument*, iDocument )->Directory();

    CEikDialog* dialog = new (ELeave) CEikFileSaveAsDialog( &filename, NULL, NULL);
    if ( dialog->ExecuteLD( R_EIK_DIALOG_FILE_SAVEAS ) ) {
      // Jos valitaan nykyinen tiedosto niin kyseessä onkin
      // tavallinen talletus
      if( STATIC_CAST( CBingoDocument*,
                       iDocument )->IsCurrentFile( filename ) ) {
        SaveBingoL();
        return;
      }
```

```
      // Jos valitaan jo olemassa oleva tiedosto, niin tuhotaan
      // se ensin
      TEntry entry;
      if( !iEikonEnv->FsSession().Entry( filename, entry ) ) {
        User::LeaveIfError( iEikonEnv->FsSession().Delete( filename ) );
      }
      // Dokumentin vastuulla on varsinainen tallettaminen
      STATIC_CAST( CBingoDocument*,
                   iDocument )->SaveModelL( filename );
      iEikonEnv->InfoMsg( R_EIK_TBUF_FILE_SAVED );
      // Päivitetään näkymä, koska nimi on muuttunut
      HandleModelChangeL();
    }
  }

  // NewBingoL
  // Uusi bingolappu
  // Mahdollisuus uuteen dimensioon ja sanastoon
  void CBingoAppUi::NewBingoL() {
    TFileName wordfile = iModel->Wordfile();
    TInt dimension = iModel->Dimension();
    CNewBingoDialog* dialog =
      new (ELeave) CNewBingoDialog( &wordfile, &dimension );
    if( dialog->ExecuteLD( R_NEW_BINGO_DIALOG ) ) {
      TRAPD( error, STATIC_CAST( CBingoDocument*,
                                 iDocument )->SetModelL( dimension,
                                                         wordfile ) );
      if( error != KErrNone ) {
        TDesC* msg =
          iEikonEnv->AllocReadResourceLC( R_BINGO_WORDFILE_ERROR );
          ErrorL( *msg );
          CleanupStack::PopAndDestroy();
      } else {
        HandleModelChangeL();
      }
    }
  }

  CBingoAppUi::CNewBingoDialog::CNewBingoDialog( TFileName* aFilename,
                                                 TInt* aDim )
    : iFilename( aFilename ), iDim( aDim ) {}

  // PreLayoutDynInitL
  // Dialogin kenttien alustaminen
  void CBingoAppUi::CNewBingoDialog::PreLayoutDynInitL() {
    SetNumberEditorValue( EBingoDimensionEditor, *iDim );
    iDriveS = (CEikDriveNameSelector*)Control( EBingoDriveNameSelector );
    iFolderS = (CEikFolderNameSelector*)Control( EBingoFolderNameSelector );
    iFolderS->SetShowSystem( ETrue );
    iFileS = (CEikFileNameSelector*)Control( EBingoFileNameSelector );

    // Riippuvuudet (tai tarkkailijat)
    iFolderS->SetFileSelectionObserver( iFileS );
    iDriveS->SetFileSelectionObserver( iFolderS );
    // Nyt pelkästään drive-kentän päivittäminen päivittää
    // folder- ja filename-kentät
```

```
  iDriveS->SetFullNameL( *iFilename );
}

// OkToExitL
// OK-nappia painettu, voidaanko sulkea dialogi?
// Päivitetään rakentajassa annetut muuttujat
TBool CBingoAppUi::CNewBingoDialog::OkToExitL( TInt aButtonId ) {
  *iDim = NumberEditorValue( EBingoDimensionEditor );
  GetFileName( iFilename, EBingoFileNameSelector );
  // Painettiinko Browse-nappia?
  if( aButtonId != EEikBidOk ) {
    // Mikäli kenttien arvo ei ole laillinen, niin
    // Käytetään alkuarvoa. Joskus dialogi toimii
    // oikein eli Browse-dialogi tekee oikean arvauksen
    // aloituspaikan suhteen. Tämä varmistus tarvitaan
    // siltä varalta, että arvausta ei jostain syystä
    // aina tehdä
    if( !iFilename->Length() ) {
      *iFilename = iDriveS->FullName();
    }
    CDirContentsListBoxModel::TSortOrder SortOrder =
      CDirContentsListBoxModel::EOrderByName;
    CEikDialog* dialog =
      new (ELeave) CEikFileBrowserDialog( *iFilename,
        CEikFileBrowserDialog::EShowSystem,
        SortOrder );
    if( dialog->ExecuteLD( R_EIK_DIALOG_FILE_BROWSE ) ) {
      iDriveS->SetFullNameL( *iFilename );
    }
    // Browse-dialogi päivittää päädialogin kentät
    // sitä sulkematta
    return EFalse;
  }
  iFileS->ValidateStateL();
  return ETrue;
}
```

Esimerkki 25. Näkymän eli napeista koostuvan bingolapun esittely.

```
// bingogame.h

#ifndef __BINGOGAME_H
#define __BINGOGAME_H

#include "bingo.h"

//
// class CBingoGame
//

class CBingoGame : public CCoeControl, public MCoeControlBrushContext,
                   MCoeControlObserver {
public:
```

```
  ~CBingoGame();
16  void ConstructL( const TRect&  aRect  );
  void SetModel( CBingoModel* model );
18
private:
20  void DestroyButtons();
  void CreateButtonsL();
22  void HandleControlEventL( CCoeControl* aControl,
                            TCoeEvent aEventType);
24  CCoeControl* ComponentControl( TInt aIndex ) const;
  TInt CountComponentControls() const;
26  void Draw(const TRect&  aRect ) const;

28  CBingoModel* iModel;
  TInt iDim;
30  CEikCommandButton*** iButtons;
  TInt* mybuttons;
32 };

34 #endif
```

Esimerkki 26. Näkymän toteutus.

```
// bingogame.cpp
2
#include <eikbutb.h>
4
#include "bingo.h"
6 #include "bingogame.h"

8 // class CBingoGame

10 // ConstructL
void CBingoGame::ConstructL( const TRect& rect ) {
12  CreateWindowL();
  iContext=this;
14  iBitmap=iEikonEnv->TexturedBitmap();
  iBrushStyle=CGraphicsContext::EPatternedBrush;
16  SetRectL( rect );
  ActivateL();
18 }

20 // Purkaja
CBingoGame::~CBingoGame() {
22  DestroyButtons();
}
24
// DestroyButtons
26 // Nappien poistaminen
void CBingoGame::DestroyButtons() {
28  for( TInt i = 0; i < iDim; i++ ) {
    for( TInt j = 0; j < iDim; j++ ) {
30      delete iButtons[i][j];
```

```
      }
      delete [] iButtons[i];
    }
    delete [] iButtons;
    iButtons = 0;
}

// CreateButtonsL
// Nappien luominen
void CBingoGame::CreateButtonsL() {
    iDim = iModel->Dimension();
    TInt width = ( Rect().Width() - iDim * 2 )  / iDim;
    TInt height = ( Rect().Height() - iDim * 2 )  / iDim;
    iButtons = new (ELeave) CEikCommandButton**[iDim];
    for( TInt x = 0; x < iDim; x++ ) {
      iButtons[x] = new (ELeave) CEikCommandButton*[iDim];
      for( TInt y = 0; y < iDim; y++ ) {
        iButtons[x][y] = new (ELeave) CEikCommandButton;
        iButtons[x][y]->SetContainerWindowL( *this );
        iButtons[x][y]->SetBehavior( EEikButtonLatches );
        iButtons[x][y]->SetTextL( iModel->Term( x, y ) );
        iButtons[x][y]->SetState( iModel->IsCrossed( x, y ) ? CEikButtonBase::ESet
                                                            : CEikButtonBase::EClear );
        iButtons[x][y]->SetExtentL( TPoint( x * width + 2 * x, y * height + 2 * y ),
                                    TSize( width, height ) );
        iButtons[x][y]->SetObserver( this );
      }
    }
}

// SetModel
// Uusi malli eli poistetaan vanhat napit ja
// luodaan uudet mallin pohjalta
void CBingoGame::SetModel( CBingoModel* model ) {
    DestroyButtons();
    iModel = model;
    CreateButtonsL();
    ActivateL();
}

// Draw
void CBingoGame::Draw(const TRect&  rect ) const {
    iEikonEnv->FillTexturedRect( rect );
}

// CountComponentControls
// Nappien lisäksi ei muita kontrolleja ole,
// joten paluuarvo on bingolapun ruutujen lukumäärä
TInt CBingoGame::CountComponentControls() const {
    return iDim * iDim;
}

// ComponentControl
// Indeksin tulkinta: x * iDim + y
CCoeControl* CBingoGame::ComponentControl( TInt index ) const {
    return iButtons[index / iDim][index % iDim];
```

```
}

// HandleControlEventL
// Kun nappia painetaan:
//   1. Päivitetään malli eli kutsutaan Toggle-jäsenfunktiota
//   2. Jos Toggle palauttaa toden, niin on tullut bingo:
//      Inaktivoidaan napit (SetDimmed)
//   3. Kerrotaan tarkkailijalle (CBingoAppUi), että nappia on
//      painettu
void CBingoGame::HandleControlEventL( CCoeControl* control, TCoeEvent event ) {
  if( event == EEventStateChanged ) {
    if ( iModel->Toggle( Index( control ) / iDim,
                         Index( control ) % iDim ) ) {
      for( TInt i = 0; i < CountComponentControls(); i++ ) {
        ComponentControl( i )->SetDimmed( ETrue );
      }
    }
    ReportEventL( event );
  }
}
```

B.3 Asennus

Esimerkki 27. Asennusviesti `note.txt`.

```
Buzz Bingo by Vespe Savikko (vespe@iki.fi)
Matti Rintala (bitti@cs.tut.fi) as "The Voice"

Word file: plain text, in Finnish
```

Julkaisija

Vuorimiehentie 5, PL 2000, 02044 VTT
Puh. (09) 4561
Faksi (09) 456 4374

Julkaisun sarja, numero ja raporttikoodi

VTT Tiedotteita 2046
VTT-TIED-2046

Tekijä(t)

Savikko, Vesa-Pekka

Nimeke

EPOC-sovellusten rakentaminen

Tiivistelmä

EPOC on Symbianin kehittämä käyttöjärjestelmä, jonka pääasiallisena kohdealustana ovat kämmentietokoneet, kuten Psion 5mx. Tällaisten laitteiden suosio perustuu pitkälti laajaan ohjelmistotarjontaan: valmiiksi asennetun perussovelluskannan lisäksi tarjolla on laaja valikoima (usein ilmaisia) kolmansien osapuolien tekemiä sovelluksia.

Sovelluskehittäjän kannalta EPOC on tietyllä tavalla kaksijakoinen: toisaalta käyttöjärjestelmä tuntuu hyvin "modernilta", mutta toisaalta jotkut yksinkertaiset asiat ovat turhan hankalia ja etenkin tarjolla oleva dokumentaatio on osin riittämätöntä.

Tämä raportti käy läpi EPOCin arkkitehtuurin yleisellä tasolla ja sen jälkeen käsittelee joitain erityispiirteitä vielä tarkemmin. Asioita tarkastellaan mahdollisimman käytännöllisellä tasolla (mm. koodiesimerkein) ja yksittäisen sovelluskehittäjän näkökulmasta, kritiikkiäkään unohtamatta.

Lopuksi rakennetaan todellinen, julkaistu EPOC-sovellus, Buzz Bingo. Lähdekoodin tarkastelun lisäksi perehdytään myös sovellusten paketointi- ja asennusmekanismeihin.

Avainsanat

EPOC architecture, EPOC applications, graphical interfaces, software development

Toimintayksikkö

VTT Elektroniikka, Sulautetut ohjelmistot, Kaitoväylä 1, PL 1100, 90571 OULU
puh. vaihde (08) 551 2111, faksi (08) 551 2320

ISBN	Projektinumero
951–38–5688–7 (nid.) 951–38–5689–5 (URL: http://www.inf.vtt.fi/pdf/)	

Julkaisuaika	Kieli	Sivuja	Hinta
Syyskuu 2000	suomi	56 s. + liitt. 36 s.	B

Projektin nimi	Toimeksiantaja(t)

Avainnimeke ja ISSN	Myynti:
VTT Tiedotteita – Meddelanden – Research Notes 1235–0605 (nid.) 1455–0865 (URL: http://www.inf.vtt.fi/pdf/)	VTT Tietopalvelu PL 2000, 02044 VTT Puh. (09) 456 4404 Faksi (09) 456 4374