TEKNILLINEN KORKEAKOULU
Sähkötekniikan osasto

Ari Mujunen

KÄSITEMALLIN JA TIETOHAKEMISTON HYVÄKSIKÄYTTÖ SOVELLUSTUOTANNON AUTOMATISOINNISSA

Diplomityö, joka on jätetty opinnäytteenä tarkastettavaksi diplomi-insinöörin tutkintoa varten Espoossa 22.05.1992

Työn valvoja

Martti Mäntylä

Työn ohjaaja

Mikko Kolehmainen

TEKNILLINEN KORKEAKOULU DIPLOMITYÖN TIIVISTELMÄ

Tekijä:	Ari Mujunen
Työn nimi:	Käsitemallin ja tietohakemiston hyväksikäyttö sovellustuotannon automatisoinnissa
Päivämäärä:	22.05.1992 Sivumäärä: 83
Osasto:	Sähkötekniikan osasto
Professuuri:	Tik-86 Tietotekniikka
Työn valvoja:	Professori Martti Mäntylä
Työn ohjaaja:	DI Mikko Kolehmainen, Tietosavo Oy

Työssä tarkastellaan uusia tapoja hyväksikäyttää entuudestaan tunnettuja menetelmiä (käsitemalli, tietohakemisto) automatisoitaessa kaupallishallinnollisten sovellusten tuotantoa ja erityisesti ylläpitoa. Työssä esitetään tapa käyttää käsitemallia sovelluksen tietokantaratkaisun kuvaamisen ohella myös varsinaisen sovellusohjelmiston rakenteen osittaiseen kuvaamiseen. Työn osana suunniteltiin tietohakemistoratkaisu, joka kykenee tallettamaan käsitemallista johdetun tietokantaratkaisun ja sovellusrakenteen ohella myös erilaisten sovelluskehitysvälineiden (eri ohjelmointikielten, käyttöliittymävälineiden, tiedonhallintajärjestelmien jne.) tarvitsemia kuvauksia käsiteltävien tietojen olemuksesta. Tämän tietohakemiston hyväksikäyttöön laadittiin työkalu, joka pystyy joustavasti käsittelemään ja tuottamaan eri sovelluskehitysvälineiden vaatimia tekstimuotoisia kuvauksia (mm. lähdeohjelmia, näyttöjen kuvauksia, tietokantojen määrittelykieliä jne.) ja lomittamaan niihin tietohakemiston sisältöä ottamatta kiinteästi kantaa tuotettavien kuvausten rakenteeseen ja syntaksiin.

AVAINSANAT: käsitemalli, ER, tietohakemisto, DD, sovellustuotanto, kaupallishallinnollinen, ohjelmisto, koodintuotto, ylläpito

HELSINKI UNIVERSITY OF TECHNOLOGY

ABSTRACT OF THE MASTER'S THESIS

Author:	Ari Mujunen
Name of the thesis:	Applying ER modelling and data dictionary techniques in software production automation
Date:	22.05.1992 Number of pages: 83
Faculty:	Faculty of Electrical Engineering
Professorship:	Tik-86 Information Technology
Supervisor:	Martti Mäntylä, Professor
Instructor:	Mikko Kolehmainen, M.Sc., Tietosavo Oy

This master's thesis discusses new ways to exploit previously known methods such as entity-relationship models and data dictionaries in business-oriented software production and maintenance automation. A way to use entity-relationship models to partially describe the structure of an application in addition to its database is presented. A particular data dictionary implementation was designed as a part of this thesis, permitting storing of data descriptions for different application development tools (such as programming languages, user interface tools, database management systems etc.) in addition to storing the ER model-based structure of an application and its database. To intelligently utilize this data dictionary a tool was realized that can flexibly process and produce textual descriptions required by different application tools (for example source code, screen descriptions, data definition languages etc.) and merge data dictionary contents to them without assuming much about their structure and syntax in a fixed way.

KEYWORDS: ER, entity, relationship, DD, data dictionary, application, business-oriented, software, code generation, maintenance

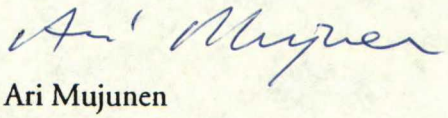
ALKULAUSE

Tämä diplomityö on tehty Tietosavo Oy:n Tuoteryhmä I:n menetelmäkehityksessä. Erityiset kiitokseni osoitan lähimmille työtovereilleni: työni ohjaajalle kehityspäällikkö Mikko Kolehmaiselle sekä projektipäällikkö Juhani Pajalalle, erikoissuunnittelija Jouni Heimoselle, suunnittelupäällikkö Hilkka Aitlahdelle ja jaospäällikkö Juhani Fominille ketään muutakaan tässä nimeltä mainitsematonta unohtamatta. Työni valvojaa professori Martti Mäntylää kiitän sekä kannustavasta kiinnostuksesta työtäni kohtaan että häneltä saamistani arvokkaista neuvoista.

TKK:n Metsähovin radiotutkimusasemaa ja sen henkilökuntaa, etenkin professori Seppo Urpoa, kiitän työni viimeistelyvaiheen tukemisesta.

Työn kirjoittamiseen käyttämästäni erinomaisesta työvälineestä haluan kiittää Apple Computer Incorporated -yhtiön lahjakkaita suunnittelijoita.[1]

Espoossa, 22.05.1992



Ari Mujunen

[1] Työn tekemiseen käytetyistä välineistä on kuvaus lähdeluettelon lopussa.

SISÄLLYSLUETTELO

SYMBOLI- JA LYHENNELUETTELO

1–4GL	First – Fourth Generation Language Ohjelmointikielten kehitysasteista käytettäviä lyhenteitä.
CASE	Computer Aided Software Engineering Tietokoneavusteinen ohjelmistotuotanto. Tietokoneohjelmistojen hyväksikäyttö toisten ohjelmistojen valmistuksessa.
COBOL	COmmon Business Oriented Language Eräs ensimmäisistä kaupallishallinnolliseen ohjelmistotuotantoon suunnitelluista ja tähän käyttöön laajalle levinneistä ns. 3GL- eli lausekielitason ohjelmointikielistä.
DAG	Directed Acyclic Graph Suunnattu silmukaton verkko. Verkkorakenne, jossa kaikilla solmujen välisillä haaroilla on suunta ja jossa juurisolmusta lähdettäessä ja haaroja pitkin edettäessä solmusta poisjohtava reitti ei tuo takaisin jo läpikäytyyn solmuun.
DB2	DataBase 2 IBM:n suurkoneympäristössä MVS-käyttöjärjestelmän alaisuudessa toimiva relaatiotiedonhallintajärjestelmä.
DD	Data Dictionary Tietohakemisto, tietoa tiedosta: varasto kuvauksia tiedon olemuksesta.
DDL	Data Description Language Tiedonkuvauskieli. Tiedonhallintajärjestelmissä käytettyjä kieliä, joiden avulla kuvataan perustettavat tietokantatiedostot tai -taulut sekä näiden sisäinen ja välinen rakenne.
DFD	Data Flow Diagram Tietovirtakaavio. SA/SD-menetelmien tapa mallittaa toimintoja tietojen virtaamisena muunnosten (transformaatioiden) läpi.
DML	Data Manipulation Language Tiedon käsittelykieli. Tiedonhallintajärjestelmissä käytettyjä kieliä, joiden avulla käsitellään (lisätään, poistetaan, muutetaan) tietokantaan säilöttyjä tietoja.

DOS — Disk Operating System
Monista levykäyttöjärjestelmistä käytetty lyhenne. Tässä työssä lyhenteellä viitataan Microsoft Corp:n Intel-suorittimille markkinoimaan ja laajalle levinneeseen MS-DOS/PC-DOS-käyttöjärjestelmään.

ER — Entity-Relationship
Yksilö-yhteys. Mallitettavan maailman mielenkiintoisten yksilötyyppien ja näiden välisten yhteyksien etsintään perustuva kuvaustapa.

ERD — Entity-Relationship Diagram
ER-kuvaustavalla piirretty kaavio, ER- eli käsitekaavio.

FORTRAN — FORmula TRANslator
Eräs ensimmäisistä ns. 3GL- eli lausetasoisista ohjelmointikielistä. Suuntautunut matemaattisten tehtävien suorittamiseen.

MVS — Multiple Virtual System
IBM:n suurkoneisiinsa (S/360-, S/370-arkkitehtuuri) markkinoima käyttöjärjestelmä.

OS/2 — Operating System/2
IBM:n ja Microsoft Corp:n yhteistyönä syntynyt Intel-suorittimille perustuviin mikrotietokoneisiin tarkoitettu graafista käyttöliittymää ja moniajoa tukeva käyttöjärjestelmä.

QBE — Query By Example
Alunperin IBM:n relaatiotietokantoihinsa kehittämä tapa kysellä tietoja täyttämällä kuvaruudulle ilmestyviin taulukkopohjiin esimerkki halutusta kyselytuloksesta.

SA — Structured Analysis
SD — Structured Design
Rakenteinen määrittely ja suunnittelu. 1970-luvulla kehitettyjä toimintojen hierarkkiseen pilkkomiseen perustuvia kuvaustapoja.

SQL — Structured Query Language
Alunperin IBM:n relaatiotietokantoihinsa kehittämä tiedon määrittely-, manipulointi- ja kyselykieli, joka muistuttaa selväkielistä englantia. Nykyisin yleisin relaatiotietokannoissa käytettävä kyselykieli, joka on myös standardoitu (ANSI, ISO).

STD	State Transition Diagram Tilakaavio. Eräissä SA/SD-menetelmän laajennuksissa käytetään tilakaavioiden havainnollistamia tilakoneita (engl. *finite state machines)* kuvaamaan, milloin ja missä tilanteissa tietovirtakaavioiden muunnokset voivat tapahtua.
VAX	Virtual Addressing Extension Digital Equipment Corporation -yhtymän tuottama minitietokoneperhe.
VMS	Virtual Memory System Digital Equipment Corporation -yhtymän VAX-minitietokoneperheeseensä tarjoama osituskäyttöjärjestelmä.

1 JOHDANTO

Ohjelmistotuotannon hitaus, virhealttius ja näistä seuraava kalleus ovat vaivanneet automaattista tietojenkäsittelyä sen syntyajoista lähtien. Tästä syystä suuri osa tietojenkäsittelyteknologian kehitysponnistuksista on kohdistunut ja kohdistuu vastaisuudessakin juuri ohjelmistokehitystyön tehostamiseen. Uuden apuvälineen tai -menetelmän keksimiseen on aina liittynyt suuria odotuksia ohjelmistotyön tuottavuuden noususta — otaksuttiinhan ensimmäisen FORTRAN-kääntäjän lähes korvaavan lukuisat konekieliohjelmoijat. Säännöllisesti on kuitenkin jouduttu toteamaan, että kertakaikkista ratkaisua tuottavuuskriisiin ei ole olemassa. Uudet oivallukset kyllä parantavat tuottavuutta, mutta eivät dramaattisesti, vaan sen sijaan lähinnä muuntavat ongelmat uusiin, ehkä helpommin hallittaviin muotoihin. Lisäksi saavutetut tehokkuusedut kuluvat nopeasti ohjelmistoille asetettujen kasvavien vaatimusten täyttämiseen.

Näistä pessimistisistä lähtökohdista katsoen lienee selvää, että tämä työ ei pyri esittelemään uutta "kokonaisratkaisua" ohjelmistotuotannon ongelmiin. Sen sijaan koetan luoda katsauksen muutamaan ehkä jo hiukan epämuodikkaaksi muuttuneeseen menetelmään: käsitemalliin, tietohakemistoon ja koodintuoton automatisointiin. Toivoakseni tehtäväni helpottaa se, että keskityn näiden apuvälineiden soveltamiseen melko rajatulla ohjelmistotuotannon alueella eli ns. kaupallishallinnollisten sovellusten tuotannossa.

"Kaupallishallinnollisilla" sovelluksilla tarkoitan ohjelmistoja, joiden tunnuspiirteinä on käsitellä sekä rakenteeltaan että tietomääriltään varsin suuria tietokantoja melko vakioituneilla tavoilla. Rakenteeltaan suuret tietokannat merkitsevät tässä yhteydessä vähintään kymmeniä, mutta tyypillisimmillään jopa satoja tietokantatauluja sisältäviä kokonaisuuksia. Ilmaisu "melko vakioituneilla käsittelytavoilla" tarkoittaa suurimman osan ohjelmistoa keskittyvän joko tietojen joustavaan arkistomaiseen käsittelyyn ("lisäys, poisto, muutos, haku") tai vaihtoehtoisesti talletettujen tietojen esittämiseen halutussa muodossa (yhteenvedot, poiminnat, ryhmitellyt tulosteet). Käsittelen kaupallishallinnollisten sovellusten luonnetta tarkemmin jaksossa 4, "Kaupallishallinnollisten sovellusten tuottamisen ja ylläpidon automatisointikeinoja".

Otsikon osa "sovellustuotannon automatisointi" viittaa siis sekä käsitemallin että tietohakemiston hyväksikäytön laajentamiseen kaupallishallinnollisten sovellusten tuotannossa ja erityisesti ylläpidossa. Painopiste on toisaalta koettaa löytää kaupallishallinnollisista sovelluksista sellaisia erityispiirteitä, jotka sallivat automaattisemman tuotannon, ja toisaalta etsiä keinoja, joilla laajentaa automaation käyttöä ohjelmiston elinkaaren loppupuolella, ylläpitovaiheessa.

Olen saanut olla mukana rakentamassa Tietosavo Oy:n energialaitoksille kehittämiä sovelluksia kolmen eri sukupolven laitearkkitehtuuriympäristöissä. 1980-luvun alkupuolella järjestelmiä valmistettiin lähinnä päätekäyttöisille pientietokoneille. Tämä ajanjakso oli voimakasta väline- ja runko-ohjelmien kehityksen aikaa ja ns. "ITU"-välineistöllä saavutettiin yllättävänkin korkeita uudelleenkäyttöasteita. 1980-luvun jälkupuoliskolla, erityisesti sen loppupuolella energialaitosten jakeluverkkojen tietojärjestelmiä kehitettiin grafiikkanäytöin varustetuille paikallisverkotetuille Unix-tyyppisille työasemille. 1990-luvulle tultaessa mikrotyöasemien verkkoa ja keskitettyä tietokantapalvelinta alustanaan käyttävien sovellussukupolvien kehitys on täydessä vauhdissa.

Se, että olen voinut seurata sovellusvälineiden ja -työskentelyn kehittymistä kolmessa erilaisessa ympäristössä, auttaa toivottavasti erottamaan sovellustuotantoongelmien joukosta sellaiset, jotka toistuvat samanlaisina käytetystä ympäristöstä riippumatta.

Työn jaksossa 2, "Peruskäsitteitä", luon lyhyen katsauksen aiheeseen liittyviin peruskäsitteisiin siten, että jokaisen peruskäsitteen kohdalla laadin myös eräänlaisen tavoitetilan ko. välineestä — tilan, johon työssä esitettävillä toimenpiteillä pyritään. Jaksossa 3, "Tietohakemiston rakenteesta", ruoditaan kaupallisia tietohakemistototeutuksia jatoimivaa toteutusta. Jakso 4, "Kaupallishallinnollisten sovellusten tuottamisen ja ylläpidon automatisointikeinoja", on omistettu kaupallishallinnollisten sovellusten laatimisessa käytettäville systeemityömalleille ja automatisoitavamman tuotantoprosessin kuvaamiselle. Jaksossa 5, "Tietohakemiston hyväksikäyttö TIP-työkalun avulla", esitellään esimerkki työkalusta, jonka avulla esimerkiksi jaksossa 3 kuvatun tyyppistä tietohakemistoa voidaan käyttää joustavasti hyväksi jakson 4 tarkoittamassa ohjelmistotuotannossa.

2 PERUSKÄSITTEITÄ

Tässä jaksossa luon lyhyen katsauksen työni otsikossa esiintyviin kolmeen peruskäsitteeseen: käsitemalliin, tietohakemistoon ja automaattiseen koodintuottoon. Näistä yksityiskohtaisimmin tarkastellaan käsitemallia; tietohakemistoa ja automaattista koodintuottoa käsitellään lyhyesti, sillä niihin palataan omissa jaksoissaan 3 ja 5.

2.1 Käsitemalli

Käsitemallintamisen (engl. *Entity-Relationship modelling)* esitteli tiettävästi ensimmäisenä Peter Chen lähteessä [Chen 1976]. Menetelmä on tarkoitettu erityisesti pysyvän tiedon mallintamiseen. Se perustuu yksinkertaisen kaavioesityksen käyttöön ja — kuten kaikki menestyksekkäät kaaviotekniikat — kykenee havainnollistamaan melko monimutkaisiakin käytännön tilanteita vähäisellä valikoimalla kaaviosymboleita.

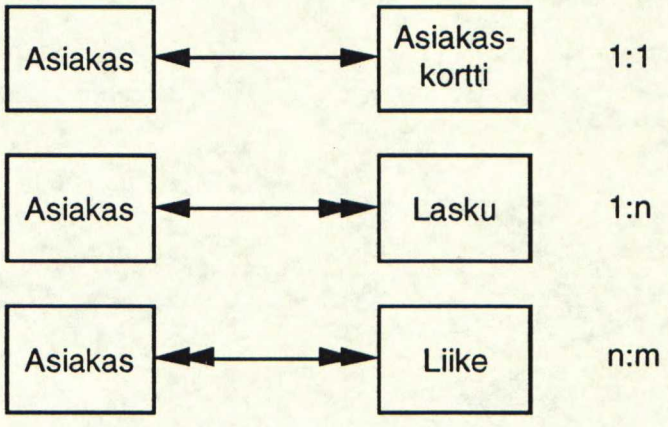
Käsitekaavioista käytetään usein suoraan englanninkielisestä lyhenteestä peräisin olevaa nimitystä "ER-kaaviot" (engl. *Entity-Relationship Diagram, ERD).*

Käsitekaavioissa on kahdenlaisia symboleita: yksilötyyppejä (engl. *entity)* ja näiden välisiä yhteyksiä (engl. *relationship).* Yksilötyypit kuvaavat mallitettavan sovellusalueen mielenkiintoisiksi katsottuja asioita: konkreettisia ja/tai abstrakteja kohteita, joista on (kuvattavan sovellusalueen mielessä) mielekästä tallettaa tietoja. Yksilötyypin symboli (nimeltään esimerkiksi "Auto") käsitekaaviossa toteaa, että mallitettavassa maailmassa on olemassa ko. tyyppisiä asioita (esimerkkiä jatkaakseni: on olemassa "Auto":ja). Yksilötyyppi siis edustaa kokonaista joukkoa ko. tyypin esiintymiä, joista jokaisesta talletetaan tietyt, mielenkiintoisiksi katsotut tiedot. Eri esiintymät erotetaan toisistaan juuri näiden talletettujen tietojen perusteella. Näitä tietoja nimitetään attribuuteiksi (engl. *attribute)* ja joissain käsitekaaviotekniikan muunnelmissa myös ne piirretään näkyviin yksilötyyppien symbolien yhteyteen.

Suomenkielisessä kielenkäytössä yksilötyyppejä nimitetään joskus lyhyyden vuoksi pelkästään yksilöiksi, vaikka tarkkaan ottaen nimitys viittaa paremminkin yksilötyypin esiintymään. Luonteva, joskin käsittääkseni vakiintumaton nimitys yksilötyypille on "kohde".

Pelkkä toteamus siitä, että on olemassa yksilötyyppejä ei vielä sisällä kovinkaan paljon käytännön tilannetta havainnollistavaa informaatiota. Tarvitaan tietoa yksilötyyppien välisistä suhteista eli yhteyksistä niiden välillä. Käsitekaavioihin tämä tieto kirjataan yhdistämällä toisiinsa liittyvät yksilötyypit yhteyssymbolein.

Kahden yksilötyypin väliselle yhteyden lajille eli kardinaliteetille (engl. *cardinality)* nähdään kolme päävaihtoehtoa: puhutaan yhdestä-yhteen-, yhdestä-moneen- ja monesta-moneen-yhteyksistä. Yksilötyyppien "A" ja "B" välisellä yhdestä-yhteen-yhteydellä tarkoitetaan sitä, että jokaista yksilötyypin "A" esiintymää kohti on tasan yksi yksilötyypin "B" esiintymä ja päinvastoin. Tällaista yhteyslajia merkitään joskus lyhyesti "1:1". Vastaavasti yhdestä-moneen-yhteys "A":n ja "B":n välillä voidaan lyhentää "1:n" ja se tarkoittaa, että jokaista yksilötyypin "A" esiintymää (esim. "A1") kohti on yksi tai useampia yksilötyypin "B" esiintymiä ja jokaista tällaista "B"-esiintymää kohti on tasan yksi "A"-tyypin esiintymä (joka on juuri mainittu esiintymä "A1"). Monesta-moneen-yhteys ("n:m") tarkoittaa, että jokaista yksilötyypin "A" esiintymää (esim. "A1") kohti on yksi tai useampia yksilötyypin "B" esiintymiä ja jokaista tällaista "B"-esiintymää kohti voi olla yksi tai useampia "A"-tyypin esiintymiä (muitakin kuin mainittu "A1").



Kuva 2–1. Yhteyslajien päävaihtoehdot.

Tällaista "pääpiirteittäistä" käsitystä tarkennetaan usein jakamalla kahden yksilötyypin ("A", "B") välinen yhteys kahteen osaan ja tarkastelemalla kumpaankin suuntaan erikseen (suunnat "A→B" ja "B→A") esiintymien lukumääriä.

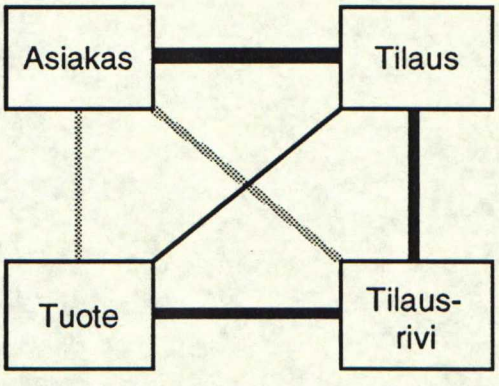
Jokaista yksilötyyppiä "X" kohti pohditaan, kuinka monta muiden yksilötyyppien esiintymää on tarpeen tallettaa kutakin "X":n esiintymää kohti. Sen sijaan, että valittaisiin "tyypillinen" esiintymien lukumäärä edustamaan yhteyttä, tarkastellaankin, montako esiintymää *vähintään* ja montako *enintään* kutakin "X":n esiintymää kohti voidaan joutua tallettamaan. Tässä yhteydessä kiinnostaviksi luvuiksi paljastuvat 0 (ei yhtään), 1 (tasan yksi) ja n (monta). Näin saadaan seu-

raavat vähimmäis- ja enimmäismäärien yhdistelmät:

Min..max	Yhteyden laji
0..0	Yhteyttä ei ole.
0..1	Yhteys tasan yhteen esiintymään voi olla.
0..n	Yhteys yhteen tai useampaan esiintymään voi olla.
1..1	Yhteys tasan yhteen esiintymään on aina olemassa.
1..n	Yhteys yhteen tai useampaan esiintymään on aina olemassa.

Taulukko 2–1. Kardinaliteettien minimi- ja maksimiyhdistelmät.

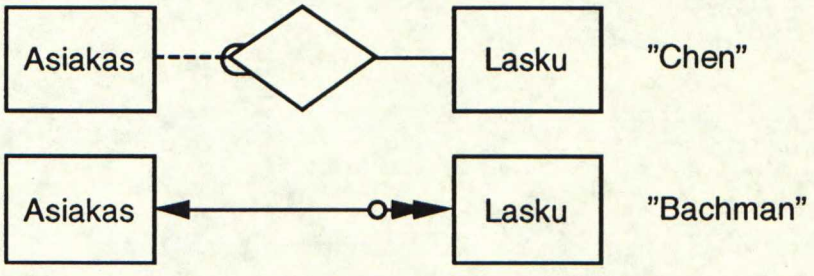
Sovelluksen kannalta kaikkien mielekkäiden yhteyksien löytämistä saattaa helpottaa ajatustapa, jossa mielletään käsitemallin kaikista yksilötyypeistä olevan yhteys kaikkiin muihin yksilötyyppeihin. Jokaisen tällaisen potentiaalisen yhteyden kohdalla pohditaan (molempiin suuntiin), mikä olisi tuon yhteyden laji. Mikäli kumpaankin suuntaan tulokseksi saadaan "0..0", voidaan yhteys jättää tarpeettomana pois ja käsitekaaviossa piirtämättä.



Kuva 2–2. Kaikkien mahdollisten yhteyksien tutkiminen.

Kuva 2–2 havainnollistaa sitä, että potentiaalinen yhteys ei ole tavallisesti yksiselitteisesti joko tarpeeton tai tarpeellinen. Kuvassa yhdisteviivojen tummuus kertoo todennäköisemmästä tarpeellisuudesta. Epätodennäköisempien yhteyksien kohdalla joudutaan ratkaisemaan, otetaanko yhteys mukaan malliin sen perusteella, kuinka runsaasti käyttöä yhteydelle voidaan ennustaa olevan mallin pohjalta luotavassa sovelluksessa.

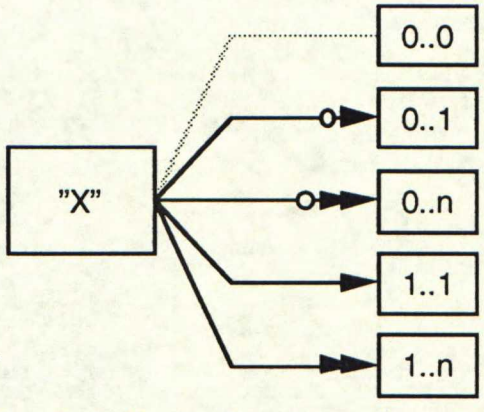
Käsitekaavioiden piirtämisessä käytetyt symboliikat eroavat toisistaan eniten juuri yhteyksien esitystavoiltaan ja siinä, kuinka yhteyksien kardinaliteetit merkitään näkyviin. Seuraavassa kuvassa 2–3 on esitetty kaksi ehkä suosituinta piirtotapaa. [Chen 1976], [Prosa 1989]



Kuva 2–3. Käsitekaavioissa käytettyjä symboleja.

Lukuisia muitakin piirtotapojen muunnelmia on olemassa. Esimerkiksi lähteessä [Chen 1981] esitetään lukuisia vivahteikkaita muunnelmia ja laajennuksia alkuperäiseen tekniikkaan — vastaavasti lähteessä [SFS 1988] on huolellisesti valittu niin yksilötyypin, yhteyden kuin attribuutinkin symbolit sellaisiksi, ettei vastaavia näytä käytetyn missään muualla.

Tässä työssä esitettävissä käsitekaavioissa käytetään jatkossa "Bachman"-tyyppistä symboliikkaa siten, että erilaiset yhteysnuolten kärjet tarkoittavat seuraavan kuvan 2–4 mukaisia kardinaliteettien minimi- ja maksimiarvoja (huomaa, että yhteyttä, jonka kardinaliteetti on "0..0", ei tavallisesti piirretä ollenkaan näkyviin):



Kuva 2–4. Kardinaliteettien merkitseminen "Bachman"-tyyppisessä käsitekaaviotekniikassa.

Aiemmin mainittiin, että joissain käsitekaaviotekniikoissa saatetaan merkitä näkyviin myös yksilötyypin symbolin yhteyteen jokaisesta tyypin esiintymästä talletettavat tiedot eli attribuutit. Menettely voi käytännössä olla hankala, sillä yksilötyypillä saattaa olla kymmeniä attribuutteja, joita kaikkia on vaikea mahduttaa kaavioon. Lisäksi juuri tietosisällöt ovat useimmin vaihtuva osa käsitekaavioon kuvattavissa olevista tiedoista.

Menestyksekkäimpiä menettelyitä ovat ne, joissa joko rajoitutaan merkitsemään näkyviin vain ns. avainattribuutit (engl. *key attributes),* joiden arvot yhdessä erottavat yksittäisen esiintymän kaikkien muiden esiintymien joukosta tai joissa attribuutit sijoitetaan luettelonomaisesti yksilötyypin symbolin sisään. [Deft 1988][2]

Tarkastellaan taulukon 2–2 muodossa kahden yksilötyypin välisen yhteyden mahdollisia erilaisia kardinaliteettiyhdistelmiä. Taulukosta kannattaa huomata, että siihen on otettu mukaan myös ne saantipolkuina mielenkiintoiset yhteyslajit, joilla ei ole joukko-opillista vastinetta: yksisuuntaiset yhteydet. Yksisuuntaisia yhteyksiä taulukossa on kaikkiaan 8 kappaletta ja ne tunnistaa siitä, että joko "A"- tai "B"-yksilötyypin kardinaliteetti on "0..0".

"A"	"B"	Yhteyden laji
0..0	0..0	Yhteyttä ei ole.
0..0	0..1	"A"→ saattaa olla yksi "B", muttei "B"→"A".
0..0	0..n	"A"→ saattaa olla useita "B", muttei "B"→"A".
0..0	1..1	"A"→ on aina yksi "B", muttei "B"→"A".
0..0	1..n	"A"→ on aina yksi/useita "B", muttei "B"→"A".
0..1	0..0	Symmetrinen.
0..1	0..1	"A"→ saattaa olla yksi "B", kuten myös "B"→"A".
0..1	0..n	"A"→ saattaa olla useita "B", mutta "B"→ saattaa olla yksi "A".
0..1	1..1	"A"→ on aina yksi "B"→ saattaa olla yksi "A".
0..1	1..n	"A"→ on aina yksi/useita "B", mutta "B"→ saattaa olla yksi "A".

[2]Tässä lähteessä kuvatussa käsitekaavioiden piirto-ohjelmassa attribuuttien näkyvyyttä tosin pystytään säätämään symbolikohtaisesti itse ja attribuutteja päivitetään ohjelmiston hallitsemien muiden kaaviotyyppien kanssa yhteiseen tietohakemistoon, jolloin attribuuttiluetteloiden muutokset heijastuvat muuallekin kuin vain pelkkään käsitekaavioon.

0..n	0..0	Symmetrinen.
0..n	0..1	Symmetrinen.
0..n	0..n	"A"→ saattaa olla useita "B", samoin "B"→"A".
0..n	1..1	"A"→ on aina yksi "B"→ saattaa olla useita "A".
0..n	1..n	"A"→ on aina yksi/useita "B", mutta "B"→ saattaa olla useita "A".
1..1	0..0	Symmetrinen.
1..1	0..1	Symmetrinen.
1..1	0..n	Symmetrinen.
1..1	1..1	"A"→ on aina yksi "B", kuten myös "B"→"A".
1..1	1..n	"A"→ on aina yksi/useita "B", mutta "B"→ on aina yksi "A".
1..n	0..0	Symmetrinen.
1..n	0..1	Symmetrinen.
1..n	0..n	Symmetrinen.
1..n	1..1	Symmetrinen.
1..n	1..n	"A"→ on aina yksi/useita "B", kuten myös "B"→"A".

Taulukko 2–2. Yhteyksien kardinaliteettiyhdistelmät.

Taulukkoa tutkimalla havaitaan, että kahden yksilötyypin välillä voi olla periaatteessa 24 erilaista yhteyttä tai ei yhteyttä ollenkaan. Näistä 24 yhteydestä vain 14 on keskenään aidosti erilaisia. Loput 10 yhteyden lajia sisältyvät jo noihin 14:n siten, että ne ovat symmetrisiä niihin nähden, mutta suunnaltaan vastakkaisia.

Vaikka useimmissa käsitekaaviotekniikan muunnelmissa kahden yksilötyypin välinen yhteys käsitetään yhdeksi asiaksi ja kuvataan yhdellä kaaviosymbolilla, havaitaan taulukon 2–2 rivien lukumäärää ja siinä esiintyvää symmetriaa tarkasteltaessa, että suunnat erottava tulkintatapa on käytännössä parempi. Erityisesti jos yhteyksiä halutaan nimetä, saattaa olla edullista ajatella yhteyden kumpikin suunta omana yhteytenään. Yhteydelle ei ole helppoa keksiä nimeä, joka olisi "totta" kumpaan tahansa suuntaan luettaessa (esimerkiksi "asiakas tilaa tuotteita" vastaan "tuotteet tilaavat asiakkaan"). Suunnat erottava ajattelu on sopusoinnussa myös yhteyksiin nimeämisen sijaan usein sovellettavan lukutavan kanssa, jossa yksilötyyppien "y1" ja "y2" välinen yhteys luetaan yleensä toteamalla, montako "y2":ta on jokaista "y1":tä kohti ja sen jälkeen, montako "y1":tä on jokaista "y2":ta kohti. Suunnan erottelua voi puolustaa myös se, että yhteyden toinen suunta on joskus "tarpeeton" (kardinaliteetti "0..0") — sitä ei tarvita tai haluta koskaan käyttää hyväksi käsitemallin pohjalta luotavassa sovelluksessa.

Toisella tapaa yhteyden kaksisuuntaisuus otetaan huomioon lähteessä [IBM2 1990] esitellyssä piirtotavassa, jossa yhteydelle merkitään näkyviin pääasiallinen suunta (engl. *primary)* ja jossa jäljellejäävä vastakkainen suunta käsitetään aina toissijaiseksi (engl. *secondary).*

Käsitemallin kestävyyttä mallinnusmenetelmänä osoittaa varmasti esimerkiksi se, että tuoreehkossa kirjassaan Peter Coad [Coad 1991] esittelee oliokeskeistä suunnittelumenetelmäänsä, jonka kaaviotekniikan ydin on johdettu suoraan käsitekaaviotekniikasta.

2.1.1 Käsitemallin käyttötapoja

Käsitemalli havainnollistaa erityisen hyvin monimutkaisia tietojoukkoja. Kaupallishallinnollisissa sovelluksissa tällainen joukko on tavallisesti jonkin tiedonhallintajärjestelmän tietokannassa oleva tiedosto- tai taulujoukko, jollaista johdannossa olleen määritelmän mukaisesti sovellukset käsittelevät melko vakioituneilla tavoilla.

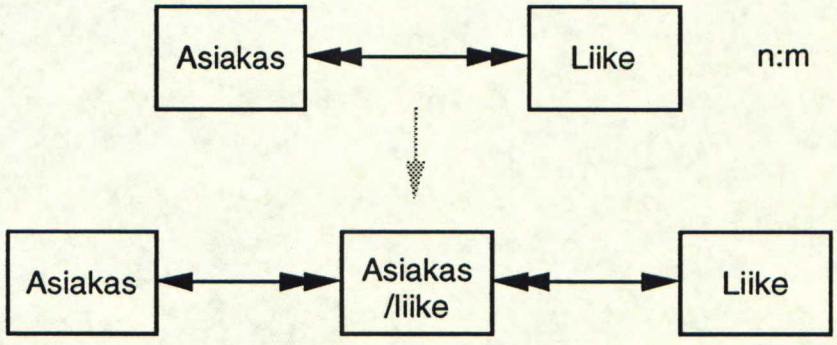
Käsitemallin avulla voidaan toki kuvata ja mallittaa muutakin kuin pysyvää ja/tai tietokannoissa sijaitsevaa tietoa. Esimerkiksi tietokoneen käyttöjärjestelmän arkkitehtuuria voidaan havainnollistaa käsitekaaviolla, jossa esitetään käyttöjärjestelmään liittyvien käsitteiden (esimerkiksi "käyttäjä, käyttölupa, laite, levylaite, hakemisto" jne.) välisiä suhteita. Käytännöllisintä käsitemallilla on kuitenkin havainnollistaa monimutkaisen tietokantaratkaisun rakennetta.

Kuinka käsitemallia ja käsitekaavioita olisi sitten parhaiten sovellettava kuvattaessa jonkin sovelluksen tietokannan rakennetta? Eräs näkökulma [Ullman 1988] suosittaa pysymistä tiukasti "käsitteellisellä" tasolla. Tällöin käsitemalliin otetaan mukaan vain sovellusalueen kiintoisat konkreettiset ja abstraktit kohteet, joiden välisiä suhteita mallitetaan mahdollisimman vivahteikkaasti käyttäen hyväksi runsaasti nimettyjä yhteyksiä sekä mahdollisesti jotain laajennettua symboliikkaa kaavioesityksissä.

Edelliselle vastakkainen lähestymistapa on sitoa käsitemallin kohteet kuvaamaan tiukasti tietokannan toteutuksen tiedostoja tai relaatiotauluja. Edelleen kaavioiden symboliikassa rajoitutaan vain perusmerkintöihin: kohteisiin ja niiden välisiin yhteyksiin. Yhteydet ovat tavallisesti nimettömiä ja monesta-moneen -yhteyksiä ei sallita, koska niitä ei voida nykytiedonhallintajärjestelmillä toteuttaa ilman välittävää tiedostoa/taulua.

Yhteydet voivat olla nimettömiä silloin, kun kahden yksilötyypin välillä vallitsee yhteys enintään yhteen kertaan. Tällöin yhteys voi saada nimensä suoraan yhdistettävien yksilötyyppien mukaan. Mikäli tietyn yksilötyyppiparin väliin tarvitaan useampia yhteyksiä, ne täytyy erottaa toisistaan nimellä, jota kutsutaan tavallisesti rooliksi (engl. *role).*

Sellaiset käsitemallit, joissa on "n:m"-yhteyksiä, voidaan muuntaa suoraviivaisesti malleiksi, joissa on vain "1:1"- ja "1:n"-yhteyksiä. Jokaista "n:m"-tyyppistä yhteyttä kohti luodaan avustava yksilötyyppi, joka tallettaa vallitsevat yhteydet apuesiintyminään. Apuyksilötyypissä on attribuutteina vähintään yhdistettävien yksilötyyppien pääavaimet vierasavaimina. Seuraavassa on kuva 2–5 muunnoksesta:



Kuva 2–5. "n:m"-yhteyden muunnos "1:n"-tyyppisiksi.

Tietokantaratkaisuun tukeutuva ja sitä havainnollistava käsitemallin käyttö ei välttämättä toimi maksimaalisena sovelluksen määrittely- ja suunnitteluvaiheiden apuna.[3] Tällaisen käyttötavan huomattavana etuna on kuitenkin mahdollisuus sitoa käsitemalli pysymään "samassa tahdissa" tietohakemiston ja todellisten, toteutettujen tietokantatiedostojen/-taulujen kanssa. Näin käsitekaavio on osa sitä kuvauskokonaisuutta (yhdessä tietohakemiston, lähdekielisten ohjelmien jne. kanssa), jota muokkaamalla sovellus muotoutuu kohti haluttua ilmiasuaan.

2.1.2 Käsitemalli ja kaupallishallinnollisen sovelluksen rakenne

Sen lisäksi, että käsitemallin avulla voidaan havainnollistaa tyypillisen kaupallishallinnollisen sovelluksen käsittelemää tietokantaa, sitä voidaan käyttää hyväksi myös muodostettaessa sovellukselle rakennetta.

---

[3]Vaiheista lisää kohdassa 4.1, "Systeemityömalleista".

Merkittävän osan tyypillisestä kaupallishallinnollisesta sovelluksesta muodostavat tietojen arkistomaiseen ylläpitoon tarkoitetut lomakkeet/ohjelmat — ohjelmat, joiden tehtävänä on hoitaa yksittäistä tiedonsyöttölomaketta. Näille ohjelmille on tunnusomaista toisaalta niiden suuri lukumäärä, toisaalta tavaton ohjelmien välinen samankaltaisuus. Kaikilta lomakkeilta halutaan hakea olemassaolevia tietoja, muuttaa ja poistaa löydettyjä sekä syöttää uusia.

Tällaisten lomakkeiden karkealle hierarkiatasolle muodostama toiminnan rakenne on syytä kuvata toiminnallisesti suuntautuneilla kuvaustekniikoilla, kuten esimerkiksi rakenteisen analyysin ja suunnittelun (engl. *Structured Analysis, SA, Structured Design, SD)* [DeMarco 1979] menetelmillä sovellusalueen hahmottamiseksi helpommin hallittaviin osiin. Sen sijaan koska useimmat arkistomaiset lomakeohjelmat eivät toiminnoiltaan juurikaan eroa toisistaan, ei ole mielekästä kuvata niistä jokaisen sisärakennetta erikseen loppuun saakka viedyiksi SA/SD-kuvauksiksi. Kuvaamalla toimintojen sijaan käsiteltävät tiedot kompaktilla tavalla keskitytään kuvauksissa juuri ohjelmasta toiseen vaihtuvaan asiaan, käsiteltäviin tietoihin. Näin vältetään tuhlaamasta energiaa samanlaisten toimintojen kuvauksien toistoon.

Tällainen käsiteltävien tietojen kuvaus voi olla esimerkiksi "ote" käsitemallista. Sovelluksen lomakeohjelmien käsittelemät tiedot kuvataan rajaamalla käsitemallista tietyn lomakkeen käsittelyn piiriin kuuluvat yksilötyypit. Rajaus aloitetaan valitsemalla jokin yksilötyyppi ohjelman/lomakkeen "pääkohteeksi" ja etenemällä tästä käsitemallin yhteyksiä pitkin ottamalla mukaan pääkohteeseen yhteydessä olevia yksilötyyppejä tarpeellinen määrä.[4]

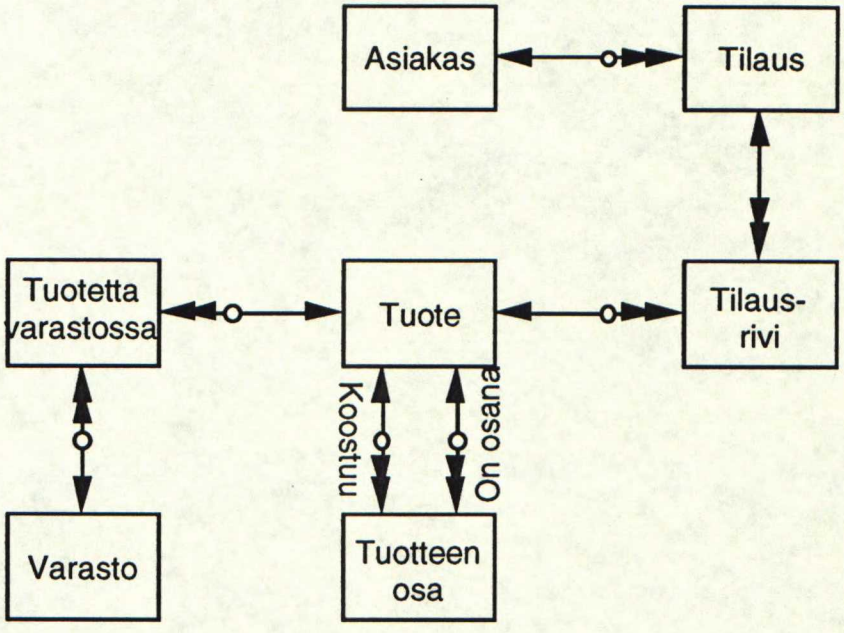
Tällainen käsitemallin ote voidaan nähdä myös puumaisena silmukattomana rakenteena (engl. *Directed Acyclic Graph, DAG).* Sen juurena on tuo mainittu pääkohde ja solmuina pääkohteeseen liittyvät ja/tai rakentuvat alikohteet, joilla kullakin voi olla vuorostaan alikohteita.[5] Silmukattomuus johtuu pääosin siitä, että nykyiset käyttöliittymätekniikat kykenevät havainnollistamaan ainoastaan sisäkkäisyyttä, jossa on kiinteä määrä sisäkkäisyystaoja. Silmukoivissa, rekursiivisissa rakenteissa sisäkkäisyystasojen määrä voi vaihdella, jolloin nykyiset käyttöliittymätekniikat joutuvat pysähtymään ennalta määrättyjen silmukointikertojen jälkeen.

---

[4]Käytän tästä kohdasta eteenpäin usein yksilötyypistä lyhyempää vastinetta "kohde", erityisesti, mikäli kysymyksessä on sellainen yksilötyyppi, jota suoraan vastaa todellinen, toteutettu tietokantatiedosto/-taulu.

[5]IBM:n Repository Manager/MVS:ssä on tätä muistuttava käsite nimeltään aggregaatti (engl. *aggregate).* Lisätietoja on kohdassa 3.2.4, "IBM Repository Manager/MVS".

Katsotaan esimerkkinä pienehköä käsitemallia ja siitä johdettua muutaman lomakemuotoisen ohjelman sovellusta. Käsitemallin aiheena on erittäin "perinteinen" kaupallishallinnollinen tilanne, tilaustenkäsittely.

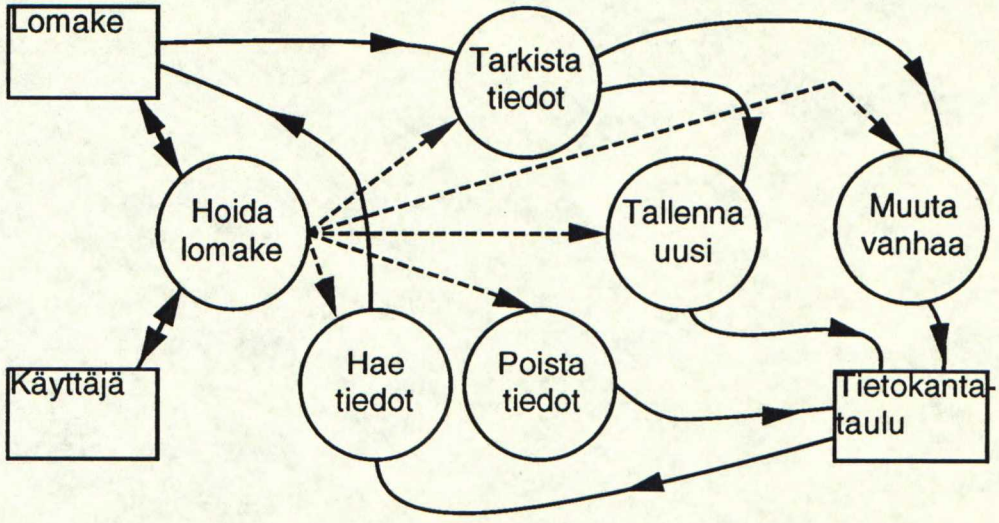


Kuva 2–6. Tilaustenkäsittelyn käsitekaavio.

Käsitemallista havaitaan, että se on laadittu edellisessä kohdassa 2.1.1, "Käsitemallin käyttötapoja" kuvattuun tietokannan toteutukseen suuntautuneeseen tapaan: esimerkiksi yksilötyyppi "Tuotetta varastossa" on eroteltu omaksi yksilötyypikseen sen sijaan, että tyyppien "Tuote" ja "Varasto" väliin olisi merkitty "n:m"-yhteys. Lisäksi voidaan nähdä, että yhteyksiä on nimetty säästeliäästi ja mieluummin on luotettu "oletuksenmukaiseen" luentaan: esimerkiksi "asiakas voi tehdä useita tilauksia, mutta kunkin tilauksen on tehnyt vain yksi tietty asiakas". Yhteyksien nimeämiseen on turvauduttu vasta siinä melko harvinaisessa tilanteessa, jossa kahden yksilötyypin välillä on useampia yhteyksiä. Tietty yksilötyyppi voi olla yhteydessä toiseen useamman kerran eri rooleissa, jolloin yhteyksien käyttötarkoitusten nimeäminen on välttämätöntä, jotta yhteydet voitaisiin erottaa toisistaan. Tässä nimenomaisessa esimerkkitapauksessa eriroolisten yhteyksien avulla tuotteet muodostavat rekursiivisen hierarkkisen rakenteen, tuotehierarkian.

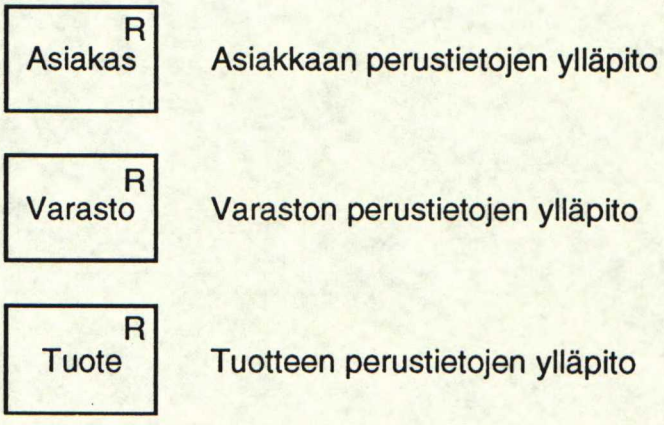
Yksinkertaisimmat lomakkeet muodostuvat asiakkaiden, varastojen ja tuotteiden perustietojen ylläpidosta. Lomakkeelle näytetään ko.yksilötyypin kaikki attribuutit ja lomakkeelta käsin voidaan perustaa uusia yksilöitä, poistaa vanhoja sekä muuttaa olemassaolevien yksilöiden tietoja. Nämä käsittelyt voidaan kuvata yhdellä kuvauksella, esimerkiksi tietovirtakaaviolla, joka on samanlainen kaikille yhtä tietokannan taulua käsitteleville lomakeohjelmille. Luonnos tällaisesta on

esitetty SA-kaaviotekniikalla[6] kuvassa 2–7.



Kuva 2–7. Lomakemuotoisen ylläpidon runko.

Tällaisen yksinkertaisen lomakkeen kuvaamiseksi kaupallishallinnollisen sovelluksen suunnitelmiin riittää todeta, mistä tietokantatauluista tällaiset sovellukset ovat tarpeen. Näin on tehty esimerkkisovelluksen osalta kuvassa 2–8. Käsitekaaviosta "lainattu" yksilötyypin symboli dokumentoi sen seikan, että ko. yksilötyyppiä vastaavasta tietokantataulusta tehdään (lähes) kaikki taulun kentät esittävä lomake ja sitä käsittelevä ohjelma vakiotoiminnoin.



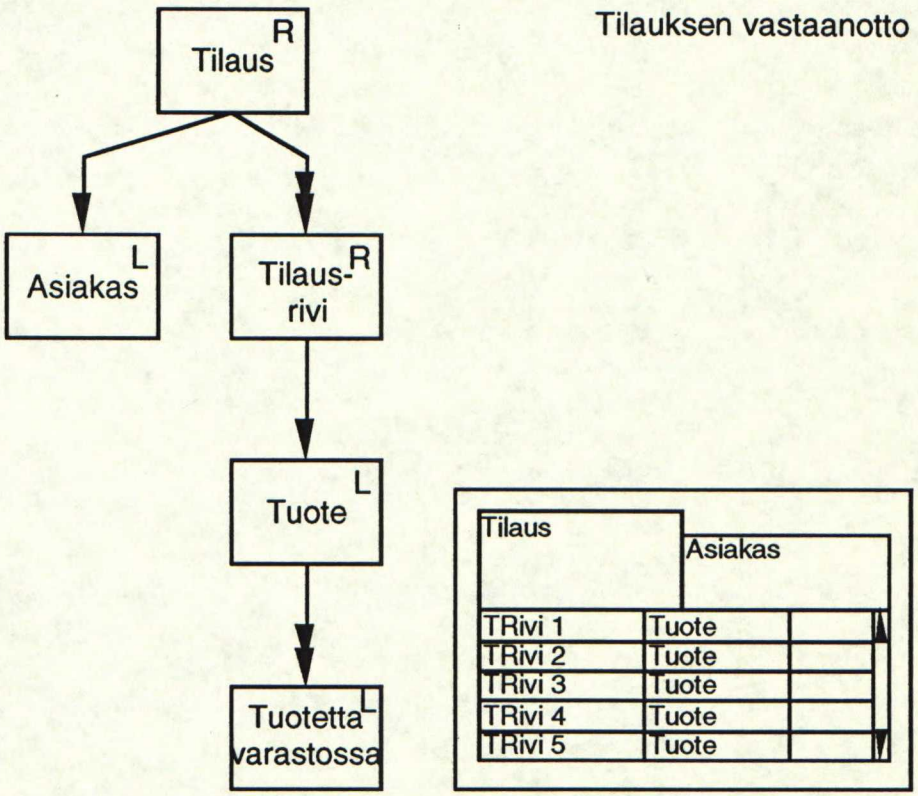
Kuva 2–8. Perustietojen ylläpidot.

Kohteiden ylänurkissa olevat kirjaimet "L" ja "R" viittaavat siihen, halutaanko ko. kohdetta tällä lomakkeella käsitellä "liittyvänä" vai "rakentuvana". "Liittyvä" lomakkeella esiintyvä kohde tuo sovellusohjelmaan lisätietoja, mutta sen tietosisältöjä ei haluta muuttaa ko. lomakkeelta. "Rakentuva" kohde puolestaan

[6]Lyhyt kuvaus SA-kaaviotekniikasta on kohdassa 3.2.1, "Structured Analysis/Structured Design -menetelmän tietohakemistot".

sisältyy osana lomakkeeseen ja sen tietosisällöt ovat vapaasti muokattavissa.

Tilaustenkäsittelysovelluksen tärkein osa on luonnollisesti se lomakeohjelma, jonka avulla tilaukset vastaanotetaan. Tätä havainnollistaa seuraava puurakenne:



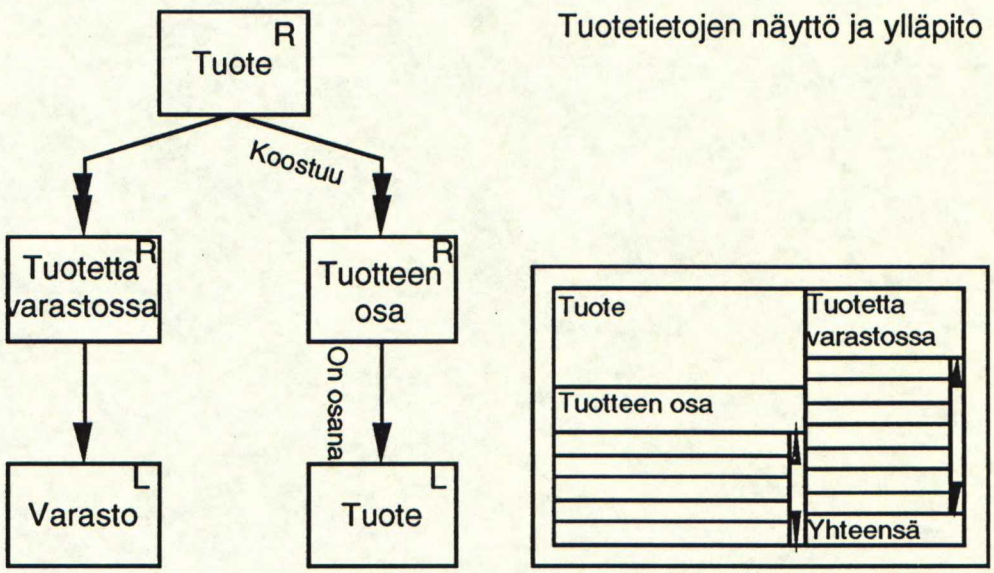
Kuva 2–9. Tilauksen vastaanotto.

Tämä esimerkki havainnoillistaa "liittyvä"/"rakentuva"-merkintöjen merkitystä tapauksessa, jossa pääkohteelle on alikohteita. Esimerkiksi asiakkaan kohdalla päätarkoituksena on liittää haluttu asiakas tehtävänä olevaan tilaukseen, jolloin kohdetta käsitellään liittyvänä eli sen tietosisältöjä ei tältä lomakkeelta ole tarkoitus muuttaa. Mikäli asiakkaan tietoja halutaan muuttaa tai vaikkapa perustaa uusi asiakas, tähän on syytä käyttää ko. tarkoitukseen tehtyä "asiakkaan perustiedot" -lomaketta — näin keskitetään taito perustaa ja muuttaa asiakkaita tilaustenkäsittelysovelluksessa yhteen paikkaan. Tilauslomakkeelta on tietenkin syytä järjestää mahdollisimman joustava pääsy asiakkaan perustietojen käsittelyyn. Tilausrivi puolestaan esiintyy tällä lomakkeella rakentuvana kohteena, sillä tilaus rakentuu siihen luotavista tilausriveistä, joiden tietosisältö määräytyy juuri tässä lomakkeessa. Tuote ja sen varastotilanne ovat lomakkeen käyttäjän kannalta mukana vain liittyvinä tietoina, vaikka ei olekaan mahdotonta, että lomakkeen taakse rakennettavaan sovellusohjelman automaattisesti tuotettuihin osiin lomitettaisiinkin käsin tehtyjä jaksoja, jotka saattaisivat päivittää vaikkapa

tuotteen varastotilanteeseen tilauksen aiheuttamat muutokset.

Tässä yhteydessä kannattaa todeta, että "liittyminen" ja "rakentuminen" eivät ole transitiivisia ominaisuuksia. Ne vaikuttavat ainoastaan siihen, kuinka ko. ominaisuudella merkityn yksilötyypin tietosisältö käyttäytyy lomakkeella ollessaan. Esimerkiksi tilauksen toimituslomakkeella saattaisi olla tilauksen vastaanoton kanssa aivan samanlainen rakenne lukuunottamatta sitä, että "Tilaus"-kohde esiintyisi lomakkeella ainoastaan liittyvänä kohteena, jolloin tilauksen toimittamisen yhteydessä voitaisiin muuttaa vain tilausrivien tietoja.

Tuotetietojen näyttöön ja ylläpitoon voidaan koota tuotteen kaikkien perustietojen ohella sekä tuotteen varastotilanne että sen mahdollinen koostuminen osatuotteista.



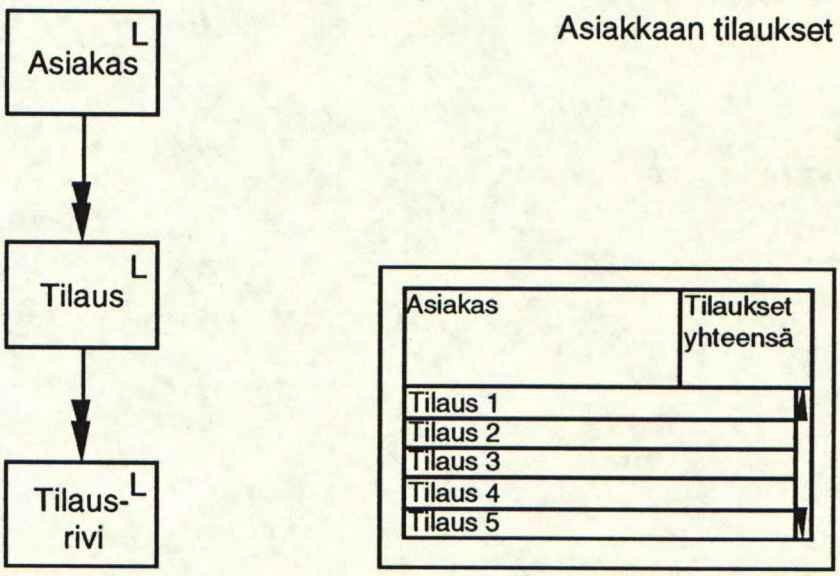
Kuva 2–10. Tuotetietojen näyttö ja ylläpito.

Tästä lomakkeesta kannattaa panna merkille saman kohteen esiintyminen useamman kerran samassa käsitemalliotteessa. Mikään ei estäisi käyttämästä myös jotain yhteyttä useita kertoja. Tuotetietoihin voitaisiin lisätä "Tuote"-kohteen alle kolmaskin haara, joka luettelisi lomakkeelle, missä toisissa tuotteissa katseltava tuote on osana ("Tuote - On osana - Tuotteen osa - Koostuu - Tuote").

Tässä tuotetiedot-esimerkissä on näytetty, kuinka käsitemalliotteen avulla voidaan käsitellä hierarkkista rakennetta sillä rajoituksella, että käsiteltävien sisäkkäisten tasojen määrä on päätettävä kiinteästi etukäteen.

Tilauksen vastaanoton aikana etsittäessä tilausriville oikeaa tuotetta esitetystä tuotetietojen näytöstä voisi olla hyötyä, joten tilausrivin tuotteen kohdalta pitäisi

järjestää helppo pääsy tähän näyttöön ja paluu takaisin. Toinen vastaava näyttö, joka voi olla kiinnostava sinällään, mutta olisi hyödyllisimmillään ollessaan helposti saatavilla tilauksen vastaanotosta käsin, on asiakkaan aiemmin tekemät tilaukset. Sen rakenne on seuraavassa kuvassa 2–11:



Kuva 2–11. Asiakkaan tilaukset.

Siirtymisten järjestämistä näytöstä toiseen käsitemallin sisältämien yhteyksien perusteella käsitellään lisää kohdassa 4.3.3, "Lomakkeiden välisten siirtymisten automaattinen järjestäminen". Tässä vaiheessa kannattaa panna merkille, että siirtymiset lomakkeesta toiseen tapahtuvat säännönmukaisesti käsitemallin yhteyksiä pitkin ja että niiden avulla voidaan välttää yksittäisen lomakkeen paisuminen "liian taitavaksi", so. osaamaan sellaista käsittelyä, joka on jo toteutettu johonkin muuhun sovelluksen osaan. Koska yhteyksien perusteella toteutetut siirtymiset pystyvät tarjoamaan kaikkia käsitemallin yhteyksien rajoissa mahdollisia jatkovaihtoehtoja, ne muodostavat sovellukselle tavallaan tiheimmän mahdollisen rakenteen.

2.2 Tietohakemisto

Esimerkiksi lähde [Martin 1976] määrittelee tietohakemiston (engl. *data dictionary)* seuraavasti: "A Data Dictionary is a repository of data about data." Suomenkielinen määritelmä ytimekkäimmillään on mielestäni seuraava:

Tietohakemisto on tietoa tiedosta.

Tyypillisesti tietohakemistoon säilötään tietoa yksittäisistä tietoelementeistä, kentistä: kentän tietotyyppi (engl. *data type),* sen ulkoasu (kuvaruudulla, paperitulosteissa jne.), käyttötarkoitus (selväkielisiä kommentteja), mahdolliset oletusarvot, rajoitteet (tiedon sallittuja arvoja rajoittavia ehtoja, engl. *constraint)* yms. Edelleen tietohakemistoon voidaan useimmiten kuvata rakenteisia tietotyyppejä, etupäässä tietueita, luettelemalla, mistä tietoelementeistä ja/tai toisista tietotyypeistä ne rakentuvat. Kolmas tavallinen tietohakemiston sisältöaihe on pysyvän tiedon, tavallisesti tietokannan tiedostojen/taulujen kuvaus.

Kuvassa 2–12 on esitetty esimerkinomaisesti, minkäkaltaisia "perinteiset" (esimerkiksi [DeMarco 1979]) tietohakemistoon talletetut yksittäisten tietoelementtien kuvaukset saattavat olla.

lähiosoite	
tyyppi	vchar(36)
muotoilu	PIC X(36)
käyttö	Katuosoitteen taltiointi
rajoite	
kehote	Lähios.
jne.	

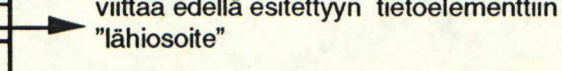
saldo	
tyyppi	fixed(9,2)
muotoilu	PIC Z(8)9.99
käyttö	Tilien, laskujen yms. mk-tilanne
rajoite	>-100.00
kehote	Saldo
jne.	

Kuva 2–12. Tietoelementtien esimerkkikuvauksia.

Rakenteisia tietoja tietohakemistoissa kuvataan seuraavankaltaisesti (kuva 2–13):

yhteystiedot

nro	kentän nimi
1	nimi
2	lähiosoite
3	postinro
4	postitoimipaikka
5	puhelin
6	telefax



Kuva 2–13. Rakenteisen tiedon esimerkkikuvaus.

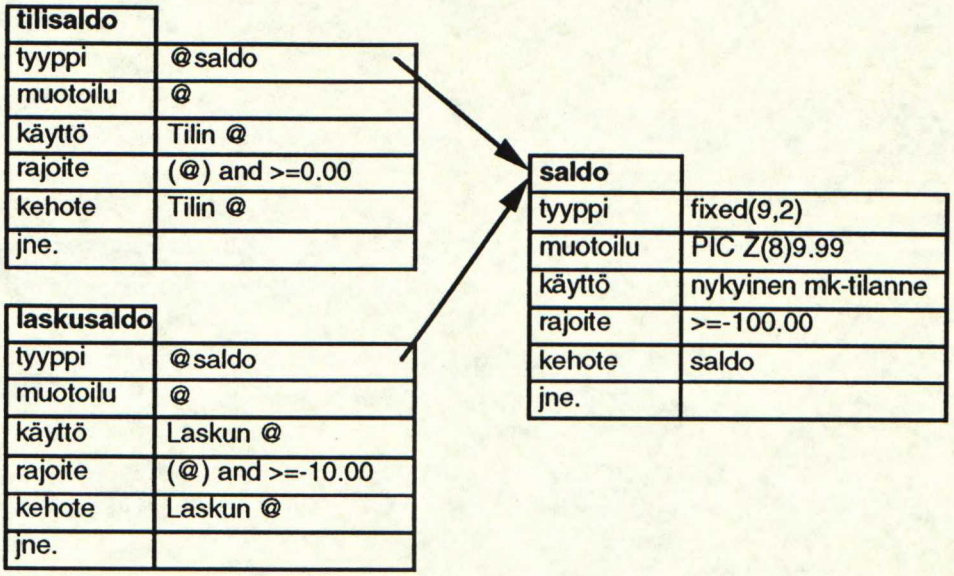
Kunkin kentän nimi on samalla viittaus toiseen tietohakemistossa kuvattuun tietoon, joka voi olla vuorostaan myös rakenteista tietoa tai yhtä hyvin perustason tietoelementti.

Luonnollisestikaan pelkkä tietojen kuvauksien säilöminen johonkin keskitettyyn paikkaan sinällään ei ole mielekästä. Mielekkyytensä tämä toiminta saa vasta säilöttyjen tietojen hyväksikäytöstä. Valitettavasti tietohakemiston käyttökelpoisuudesta mieleen tulevia käsityksiä leimaa alkuaikojen tavallisin käyttötarkoitus: sovellusten määrittely- ja suunnitteludokumenttien ("systeemisuunnitelmien") osana toimiminen. Tällaisilla tietokuvauksilla oli taipumus jäädä suunnitteluvaiheen reliikiksi: koska tietohakemistoon suunniteltaessa vietyjä tietokuvauksia ei voitu käyttää toteutustyössä mitenkään koneellisesti hyväksi, ei niitä jaksettu pitää ajan tasalla toteutettaessa sovellusta — sikäli, kun niitä edes alunperin laadittiin loppuun asti ennen toteutukseen ryhtymistä. Koneellistettuja tietohakemistoteteutuksia, jotka palvelevat lähinnä vain omaa olemassaoloaan, kuvailee skeptiseen sävyyn jo DeMarco kirjassaan vuodelta 1979. [DeMarco 1979]

Tietohakemiston käytön merkittävin etu on tiedon olemuksen kuvausten *keskittyminen* yhteen paikkaan helpommin ylläpidettäväksi. Keskittymistä voidaan tunnistaa kahta lajia: keskittymistä eri sovelluskehitysvälineiden kuvauksista (esimerkiksi lähdekielisistä ohjelmista, näyttöjen kuvauksista, tietokannan määrittely- ja käsittelykielisistä (engl. *DDL, Data Description Language, DML, Data Manipulation Language)* sovellusosista yms.) kohti tietohakemistoa ja toisaalta tietohakemiston sisäistä kuvausten keskittymistä.

Eräs mahdollisuus mieltää eri sovelluskehitysvälineiden kuvausten keskittyminen tietohakemistoon on ajatella tietohakemistoa eräänlaisena pysyvänä ja sovelluskehitysvälineiden läpi yhteisenä käsiteltävien tietojen tyyppikuvausten taltiointipaikkana, kaikkien välineiden käyttämänä "suursymbolitauluna" (vrt. ohjelmointikielen kääntäjän symbolitaulu).

Tietohakemiston sisäistä kuvausten keskittämistä havainnollistan esimerkillä kuvassa 2–14. Sinänsä on merkillistä todeta, että varsin harvassa nykyisessä käytännön tietohakemistossa on toteutettu näinkin ilmeinen keskittämiskohde, kuten jaksossa 3.2, "Olemassaolevia tietohakemistototeutuksia", tullaan havaitsemaan.



Kuva 2–14. Kuvausten keskittäminen tietohakemiston sisällä.

Esimerkissä on kuvattu kaksi uutta tietoelementtiä "tilisaldo" ja "laskusaldo", jotka molemmat perustuvat yhteiseen "saldo"-elementin kuvaukseen. "@"-merkein[7] on ilmaistu, että ko. ominaisuus lainataan yhteiseltä kantaelementiltä. Tällaiset "lainausmerkinnät" talletetaan tietohakemiston sisällöksi ja ne näkyvät tietohakemistoa ylläpidettäessä, mutta tietohakemistoa *hyväksikäytettäessä* merkinnät lavennetaan auki. Nyt esimerkiksi kysyttäessä tietohakemistosta "laskusaldo"-tiedon käyttöä vastaukseksi saadaan "Laskun nykyinen mk-tilanne". Näin "saldo":on kohdistuvat muutokset näkyvät kaikkien "saldo":n käyttöpaikkojen lisäksi myös "tilisaldo":n ja "laskusaldo":n käyttöpaikoissa. Tällainen menettely ei tietenkään ole periaatteeltaan mitenkään uusi ja ainutlaatuinen: useiden ohjelmointikielten (esimerkiksi Ada [Ada 1983]) tyyppimäärittelyt antavat mahdollisuuden määritellä uusia tietotyyppejä aiemmin määriteltyjen tietotyyppien pohjalta siten, että uuden tyypin kuvauksessa annetaan vain kantatyyppiin nähden halutut muutokset.

Ulkoisen, sovellusvälineiden välisen keskittämistavoitteen saavuttamiseksi tietohakemiston pitää kyetä tallettamaan monien erilaisten kehitysvälineiden tarvitsemia tietoja tiedon rakenteesta ja toisaalta samanaikaisesti minimoida näiden tietojen monistuminen tietohakemiston sisällä. Näitä tavoitteita käsitellään tarkemmin jaksossa 3, "Tietohakemiston rakenteesta".

Vetoavilta vaikuttavista eduista huolimatta tietohakemistot eivät ole vieläkään levinneet erityisen laajaan käyttöön kaupallishallinnollisessa ohjelmistotuotan-

[7] "@"-merkintä on valittu melko mielivaltaisesti edustamaan periaatetta, eikä sitä tule käsittää minään suositeltuna syntaksina.

nossa. Tietohakemistoja on tosin melko laajassa käytössä osana relaatiotiedonhallintajärjestelmiä, CASE-työkaluja ja 4GL-kehittimiä, mutta tällaiset yhden välineen "omat" tietohakemistot harvoin tarjoavat mahdollisuutta käyttää niihin talletettuja tietokuvauksia missään muissa sovelluskehitysvälineissä. Tämäntapaisista käytännön toteutuksista esitellään muutama kohdassa 3.2, "Olemassaolevia tietohakemistototeutuksia".

Yksi harvoista vielä vielä nykyään menossa olevista pyrkimyksistä liittää tietohakemisto merkittäväksi osaksi sovelluskehitystä on IBM:n AD/Cycle-sovelluskehityspuitteistoon kuuluva "Repository Manager/MVS". [IBM1 1991] Tätä laajamittaista IBM:n "kuvauskannaksi" tietohakemiston sijaan nimittämää toteutusta käsitellään lyhyesti kohdassa 3.2.4, "IBM Repository Manager/MVS".

2.2.1 Tietohakemisto ja oliokeskeinen ajattelu

Ehkä osaselityksiksi tietohakemistoratkaisujen suosion laskulle viime aikoina voidaan laskea kuluneen runsaan vuosikymmenen aikana runsaasti huomiota saaneet tiedon abstrahointi, [Liskov 1981] kätkentä/kapselointi ja — erityisesti viime vuosina — oliokeskeinen ajattelu. [Kim 1989], [Meyer 1988] Näissä lähestymistavoissa on keskeisellä sijalla pyrkiä kätkemään käsiteltyjen tietoolioiden todellinen tietosisältö olioiden käyttäjiltä. Tietohakemistohan pyrkii tavallaan tällaiselle kätkennälle vastakkaiseen päämäärään: tiedon rakenteen kuvauksen mahdollisimman helppoon saatavuuteen mahdollisimman monesta paikasta käsin. Ovatko abstraktit tietotyypit ja oliot sitten tehneet tietohakemiston tarpeettomaksi?

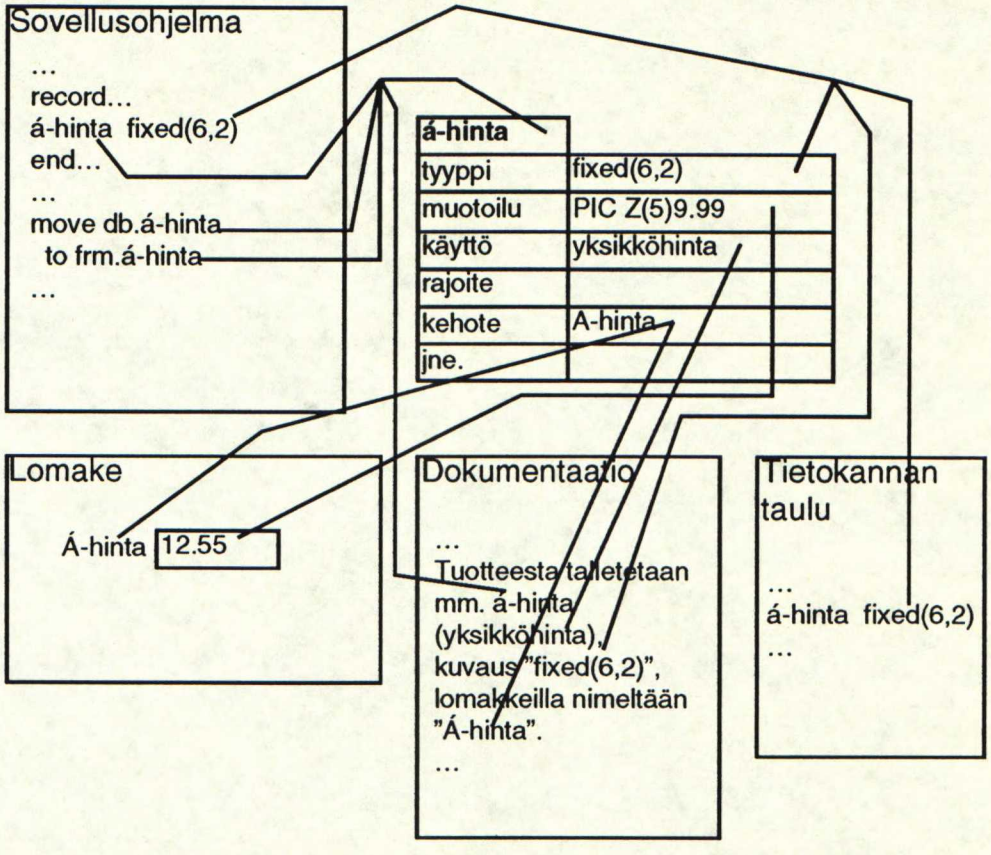
Tiedon abstrahointi ja oliopohjainen käsittely ovat oivallisia välineitä hallita monimutkaistuvia ohjelmistoja. Niiden käyttökelpoisuus esimerkiksi suorakäyttöisten käyttöliittymien laatimisessa on kiistaton. [Borland 1990] On todennäköistä, että esimerkiksi graafisen käyttöliittymän toteuttaminen kaupallishallinnolliseen sovellukseen ei tee tässä suhteessa minkäänlaista poikkeusta. Sen sijaan itse kaupallishallinnollisten, sovelluksen "omien" tietokohteiden suhteen en ole ollenkaan varma, onko kohteiden tietojen kätkeminen sovelluksen tavoitteiden mukaista. Väitän, että kaupallishallinnollisen sovelluksen varsinaisena tarkoituksena on julkistaa suuri yhteinen tietovarasto laajan käyttäjäjoukon tutkittavaksi ja käsiteltäväksi. Esimerkiksi haluttujen tietojen etsintä tapahtuu tavallisesti sen perusteella, että talletettujen yksilöiden tietosisällöt tunnetaan ja etsintään käytetään tietosisältöjen arvojen varaan rakennettuja hakuehtoja. Ellei tietosisältöjä tunneta, hakuehtojen ohella esimerkiksi tulosteiden sisällön ja ulkoasun laatiminen vaikeutuu kovin.

Tämä ei tarkoita, etteikö esimerkiksi kutakin käsitemallin yksilötyyppiä kohti olisi mielekästä luoda olio-ohjelmointikieltä käytettäessä vastaavaa luokkaa. On olemassa operaatioita, jotka sopivat luokan metodeiksi, koska ne operoivat pelkästään luokan omilla instanssimuuttujilla. Toisaalta on syytä olla odottamatta tästä osittaisesta kapseloinnista valtavia etuja, sillä keskeisimmät näiden olioiden käyttötavat murtavat kapseloinnin melko suoraviivaisesti: se, että yksilöä edustavan olion (lähes) kaikki instanssimuuttujat halutaan lomakkeelle muutoksia varten, ei muutu muuksi, vaikka muuttujia kuinka kapseloitaisiin arvon palauttavien ja sitä muuttavien metodien sisään. Uuden muuttujan lisäys vaikuttaa lomakkeeseen ja osaan ohjelmakoodia ja tässä yhteydessä kapselointifunktioparin lisäämisestä on tuskin muuta vaikutusta kuin työtä lisäävää haittaa. Tämä kapseloinnin väistämätön murtuminen havainnollistuu kohdan 4.3.1, "Tietosisällöistä riippuvat automatisoinnit" kuvassa 4–4, jossa näytetään, kuinka kaupallishallinnollisen sovelluksen keskeisin toiminnallisuus on juuri yksittäisten (instanssimuuttujatasoisten) tietojen uudelleenjärjesteleminen.

Tiivistäisin väitteeni pohdittavaksi seuraavaan kysymykseen: "Onko edullisempaa kätkeä tyypillisen tietokannassa sijaitsevan kaupallishallinnollisen kohteen tietosisältö vai julkistaa se kaikille eri sovelluskehitystyökaluille, jotta ne voisivat mahdollisimman automaattisesti adaptoitua tietosisällön muutoksiin?" Suosittaessani käsitemalliin pohjaavan tietohakemiston käyttöä päädyn julkistamisen kannalle.

2.2.2 Tietohakemisto ja kaupallishallinnollisen sovelluksen rakenne

Havaitsimme kohdassa 2.1.2, "Käsitemalli ja kaupallishallinnollisen sovelluksen rakenne", että käsitemallin sisältämä tietosisältö voidaan saada kiinteään yhteyteen kaupallishallinnollisen sovelluksen rakenteeseen. Tietohakemisto maksimoi tämän suhteen toimimalla jaettuna sijoituspaikkana käsitemallin tiedoille ja täydentämällä niitä yksittäisen kohteen sellaisilla tiedoilla, jotka on tarkoitettu yhteiskäyttöisiksi usean sovelluskehitysvälineen kesken. Seuraava kuva 2–15 havainnollistaa, kuinka useaan paikkaan jo yhden yksittäisen tietoelementin kuvauksesta joudutaan monistamaan tietoa, vaikka kyseessä olisi vain yhden sovelluslomakkeen tekeminen. On helppo nähdä, millaisia työsäästöjä on saavutettavissa, mikäli tämä monistaminen onnistuu jokaiseen tarvittuun käyttöpaikkaan koneellisesti ja vielä siten, että tietohakemiston sisällön muuttuessa monistus voidaan tehdä uudelleen myöskin ilman käsityötä.



Kuva 2–15. Kuvausten keskittäminen eri sovelluskehitysvälineiden kesken.

Esimerkissämme voisimme tarkastella vaikkapa tilannetta, jossa valittu tietotyyppi "fixed(6,2)" osoittautuu liian pieneksi tallettamaan kaikkein suurimpia yksikköhintoja. Kentän pidentäminen vaikuttaa suoraan vähintään kolmeen käyttöpaikkaan: tietokannan taulun kuvaukseen, sovellusohjelman käyttämään tietuekuvaukseen ja dokumentaatioon. Välillisesti muutos heijastuu vielä pidemmälle, sillä myös kentän muotoilua on muutettava pidemmäksi, jolloin korjattavaksi tulee vielä sovellusohjelman käyttämä lomake. Myös tietokannassa taulun kentän pidentäminen vaatii tavallisesti useimmissa tiedonhallintajärjestelmissä muunnostoimenpiteitä. Esimerkin tilanne muuttuu erityisen todentuntuiseksi, kun muistetaan, että todellisessa sovelluksessa on todennäköisesti á-hintatyyppisiä kenttiä useissa tietokantatauluissa, joista jokainen esiintyy kymmenissä sovellusohjelmissa ja niitä vastaavissa lomakkeissa ja tulostemalleissa. Tietenkin "á-hinta" esiintyy kymmenissä paikoissa myös sovelluksen dokumentaatiossa ja käyttöohjeissa. Á-hintoja sisältäviä tietokantatauluja on käytössä kaikilla saman sovelluksen käyttäjäasiakkailla, joille jokaiselle on järjestettävä em. tietokannan muunnostoimenpiteet.

Tällaista esimerkkiä tutkittaessa tuntuu oudolta, että kuvatunlaisen monistus- ja korjailutyön käsin tekemistä ei pidetä kovinkaan kummallisena tämän päivän kaupallishallinnollisessa ohjelmistotuotannossa. Käsityöhön on jouduttu mm. kaupallisten tietohakemiston sisältävien välineiden osoittauduttua käytössä pettymyksiksi kyvyttömyydessään auttaa monen sovelluskehitysvälineen kesken tehtävien muutosten hallinnassa ja käytettävien välineiden lukumäärän jatkuvasti kasvaessa.

2.3 Automaattinen koodintuotto

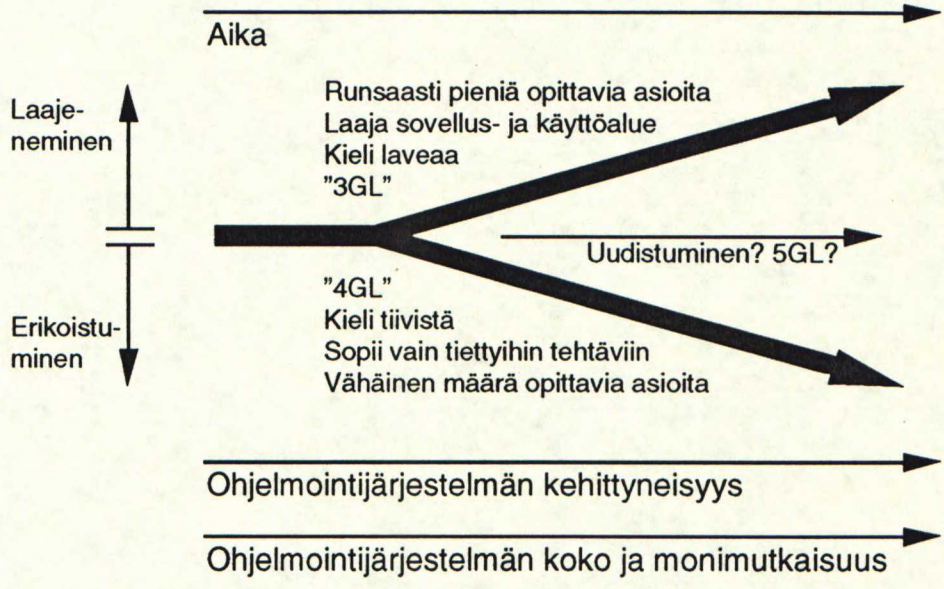
Automaattisella koodintuotolla tarkoitetaan laajasti ymmärrettynä kaikkia menetelmiä, joissa tietokone sopivan ohjelmiston avulla avustaa sovellusohjelmakoodin tuottamisessa. Näin laajaan määritelmään sopivat jopa erilaiset lähdetekstin muokkaamiseen käytetyt tekstintoimittimet; apuvälineiden joukkoa voikin ryhmitellä niiden "automaattisuusasteen" mukaan: toisessa ääripäässä ovat tällöin lähes ilman ihmistyötä toimivat automaattiset ohjelmointikielten kääntäjät.

Juuri ohjelmointikielten kääntäjien alueella apuvälineiden kehityshistoria ulottuu pitkälle koko automaattisen tietojenkäsittelyn alkuhetkiin. Usein viitataan ohjelmointikielten sukupolviin: 1., 2., 3. ja 4. sukupolven kieliin (engl. *1st, 2nd, 3rd, and 4th generation languages, 1–4GL).* 1. sukupolvella viitataan konekieliohjelmointiin, 2. sukupolvella assembler-tasoiseen konekieliohjelmointiin ja 3. sukupolvella lausekielitasoisiin, proseduraaliseen ja/tai funktionaaliseen ohjelmointiin perustuviin kieliin. [Friedman 1991] Neljännen sukupolven kielten tunnusmerkeistä ei enää ollakaan kovin yksimielisiä. Yhteisenä piirteenä kielille, joita niiden kehittäjät kutsuvat nimityksellä "4GL", on kuitenkin pyrkimys tehtävänläheisyyteen: kielet sisältävät jollekin sovellusalueelle sopivia voimakkaita rakenteita eivätkä pyri sopimaan kaikenlaisten tehtävien ratkaisuun.

Jos kehitys konekieliohjelmoinnista lausekieliin olikin nopeaa, kuten voidaan päätellä esimerkiksi FORTRAN-kielen kehittämisen sijoittumisesta ajallisesti lähelle ensimmäisten tietokoneiden kehittämistä, [Backus 1981] voisi ulkopuolinen tarkkailija kuitenkin helposti arvostella myöhempää kielikehitystä "pysähtyneeksi". Tästä on osoituksena mm. se, että jatkuvasti kehitetään uusia kolmannen sukupolven kieliä, esimerkiksi "Oberon". [Wirth 1988] Samoin seuraavaa, viidettä kielisukupolvea ei ole näköpiirissä muualla kuin rohkeimpien 4GL-kielten tuottajien myyntiesitteissä.

Ehkä selityksenä tähän ilmiöön voisi olla ohjelmointijärjestelmien kehittyminen uusien periaatteiden käyttöönoton (kuten juuri lausekieliperiaate oli) sijaan lähin-

nä kooltaan ja monimutkaisuusasteeltaan yhä suuremmiksi. Ohjelmointikielen kehittämiseen ja toteuttamiseen käytetty kasvava panos kanavoituu joko yhä "suuremman" 3GL:n tai yhä "taitavamman" 4GL:n suuntaan. Tätä havainnollistan kuvassa 2–16.



Kuva 2–16. Ohjelmointijärjestelmien kehityssuuntia.

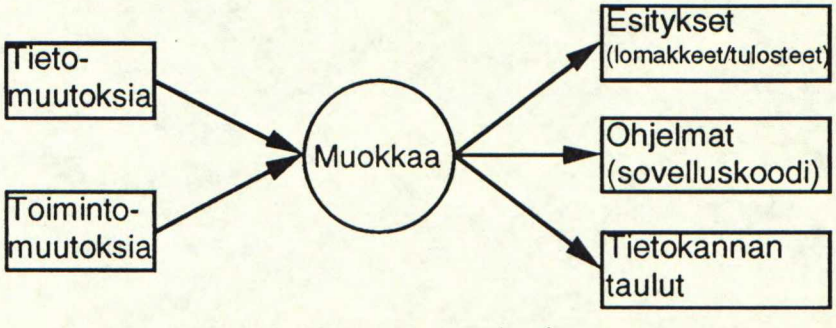
Klassikoksi muodostuneessa kirjassaan "Application Development without Programmers" [Martin 1982] James Martin näkee sovellusten loppukäyttäjät hyväksikäyttämättömänä sovelluskehitysresurssina, joka vain odottaa 4GL-tyyppisiä, erikoistuneita ja voimakkaita välineitä, joiden avulla he voivat itse luoda tarvitsemiaan käsittelykäytäntöjä yrityksen kaupallishallinnollisiin tietokantoihin. Vaikka monet Martinin esittämistä tulevaisuudennäkymistä (kuten esimerkiksi keskusyksiköiden suorituskyvyn vähintään kymmenkertaistuminen vuosina 1982–1992) ovatkin toteutuneet ja osittain jopa ylittyneet, valtaosan sovelluksista tekevät edelleen ohjelmistoalan ammattilaiset — ehkä kuitenkin yhä vähenevässä määrin "ohjelmoijat". Nykyään käytettävät sovelluskehitysvälineet ylittävät laajuudeltaan moninkertaisesti eräkäsittelymuotoisen COBOL-ohjelmoinnin vaatiman tietotaitotason, jota Martin käyttää kirjassaan toistuvasti "ohjelmoinnin" näköispatsaana. Nämä välineet ovat kuitenkin suhteellisen harvoin pitkälle erikoistuneita 4GL-tyyppisiä järjestelmiä, vaan paljon useammin lukumääräisesti laajeneva joukko yhdessä käytettäviksi tarkoitettuja laaja-alaisia 3GL-luonteisia sovellusvälineitä.

Miksi kehitys on sittenkin monelta osin kulkenut 3GL-tyyppisten järjestelmien laajenemisen suuntaan vastoin Martinin ennusteita? Erämuotoinen käsittely ja ehkä COBOL:nkin käyttö on vähentynyt, mutta niitä ei ole laajamittaisesti

syrjäytetty 4GL-tyyppisin välinein. Saattaa olla, että Martin oletti ennusteitaan tehdessään käyttäjien tyytyvän sovelluksiin, jollaisia saadaan aikaiseksi erämuotoisella COBOL:lla — tällaisiin käyttötarkoituksiin 4GL-tyyppiset välineet ovat ehkä jossain määrin levinneet. Loppukäyttäjät haluavat sovelluksiin kuitenkin ominaisuuksia, joita 4GL- ja QBE-tyyppiset välineet eivät kykene tarjoamaan — halutaan graafisia käyttöliittymiä heterogeenisessä työasemaverkossa, monimutkaisia relaatiotietokantaan kohdistuvia päivityksiä lyhyin vasteajoin toteutuvana tapahtumankäsittelynä jne. Tällaisten toteuttamisesta on tullut aikamme "COBOL-ohjelmointia": sovellusten laatijat joutuvat käyttämään uusia, laajoja, alati kehittyviä, runsasta opettelua vaativia ja hyötysuhteeltaan matalia sovelluskehitysvälineitä.

Loppukäyttäjät ovat tällaisten uusien välineiden samassa tilanteessa kuin 1980-luvun alussa COBOL-ohjelmoinnin kanssa. Heidän kannaltaan ne ovat liiallista erikoistumista ja opettelua vaativia. Saattaa myös olla, että loppukäyttäjien valjastaminen sovelluskehittäjiksi tuottaisi sovelluksen rakenteen hallintaongelmia: organisaatioon syntyy joukoittain "miniatyyriosasovelluksia", jotka eivät keskustele toistensa kanssa ja jotka ovat ominaisuuksiltaan ehkä päällekkäisiä. Käyttäjät turhautuvat tuottaessaan itselleen sovelluksia, joita eivät oikeastaan halua käyttää, koska niistä puuttuu piirteitä, joita he pitävät tärkeinä. Joudumme ehkä sittenkin etsimään keinoja sovelluskehittäjien tuottavuuden moninkertaistamiseen.

Eräs keskeisistä keinoista nostaa sovellustyön tuottavuutta parempien suunnittelu- ja projektinohjausmenetelmien ohella on nostaa sovelluskoodin tuottamisen automaatioastetta. Tavallisimmin nähty periaate entistä automaattisemman koodintuoton mahdollistavan sovelluskehitysvälineen laadinnassa on luoda kokonaan uusi ohjelmointikieli. Harvemmin esiintyy yrityksiä valjastaa käytössä olevia välineitä tehokkaampaan yhteiskäyttöön niitä sitovalla järjestelmällä. Tällaisen järjestelmän luomiseksi olen koettanut jakaa kaupallishallinnollisen sovelluskehityksen kehitystyötä osiin lähtökohtana nykyisessä työskentelyssä pullonkaulaksi olettamani toistuva uudelleentekeminen.



Kuva 2–17. Perinteinen koodintuotto.

Kuvassa 2–17 havainnollistan, kuinka tavanomaisessa sovellustyöskentelyssä esille tulevat muokkaustarpeet toteutetaan käsityönä suoraan perinteisten sovellusvälineiden lähdekielisiin kuvauksiin. Taito, kuinka tämä tehdään, on muokkaustyötä tekevän henkilön ammattitaitona. Tästä syystä muokkausten tekemiseen vaaditaan aina ammattitaitoinen henkilö, jonka on uhrattava työpanostaan oppimiensa mallien toistuvaan toteuttamiseen.

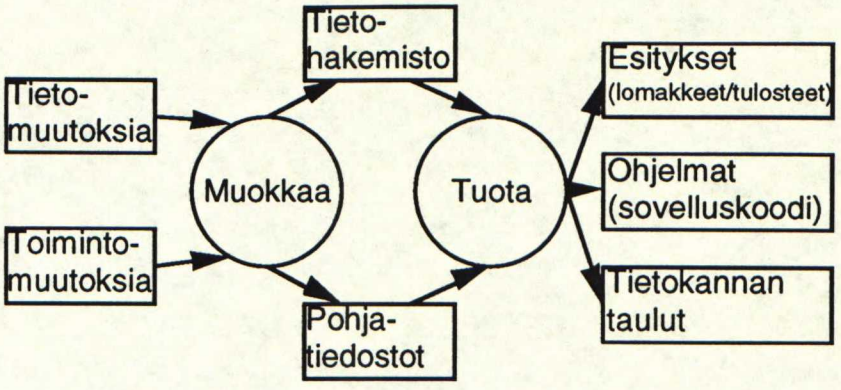
Kaupallishallinnollisen sovelluksen tiiveimmin muuttuva osa on sen käsittelemien tietojen kuvaus.[8] Samalla se on laajuudeltaan kertaluokkia tiiviimpi kuin varsinainen sovellus ja näin ollen tehokkaammin ja nopeammin ihmistyönä muokattavissa. Tiedon kuvauksen taipumuksena on hajautua kaikkien käytettyjen sovellusvälineiden tarvitsemiin kuvauksiin ja vaikuttaa niihin. Niinpä tietojen kuvaus kannattaa koettaa erottaa tietohakemistoon siten, että kaikki (tai ainakin mahdollisimman monet) kuvauksista riippuvat kohdat sovelluksen lopputulosmateriaaleissa syntyisivät koneellisen lomituksen tuloksena.

Toimintojen puolella voidaan käyttää hyväksi kaupallishallinnollisen sovelluksen ohjelmaosien keskinäistä samankaltaisuutta: voidaan löytää suppea joukko perustoimintoja, joita halutaan toteutettavaksi vaihtuville tietojoukoille. Kun tällä tavoin varotaan monistamasta samaa toimintaa käsin useaan paikkaan vain odottamaan toimintojen muuttumista, saavutetaan merkittäviä ylläpitoetuja.

Sen sijaan, että sovellusammattilainen toteuttaisi suoraan jonkin toiminnon muokkaamalla joukkoa lähdetiedostoja, hänen kannattaa siirtyä taltiomaan, miten hän aikoi muokkaukset suorittaa. Tässä prosessissa pyritään etsimään uusia perustoimintoja (so. toimintoja, joita voitaisiin soveltaa muissakin tapauksissa kuin juuri sillä hetkellä ajankohtaisessa tilanteessa) ja löytämään toteutettavasta toiminnosta tietojen kuvauksista riippuvat osat. Sekä uudet perustoiminnot että muissa toiminnoissa tietojen kuvauksista riippuvat osat kuvataan välineelle, joka pystyy lomittamaan tietohakemistossa olevia tietoja käytetyn sovellusvälineen kuvauksen joukkoon. Tällä periaatteella saadaan talletettua merkittävä osa siitä taidosta, jota olisi käytetty, mikäli muokkaus olisi tehty käsityönä, sellaiseen muotoon, että sovelluksen lopputulosmateriaalit on mahdollista tuottaa koneellisesti uudelleen minkä tahansa tekemiseen liittyvän osatekijän muuttuessa. Näin voidaan kompensoida käytettävistä sovellusvälineistä mahdollisesti puuttuvia kykyjä adaptoitua tietosisältöjen muutoksiin tai puutteellisia mahdollisuuksia osittaa toiminnot uudelleenkäytettäviin jaksoihin.

---

[8]Kauapllishallinnollisen sovelluksen tuotantoprosessia käsitellään tarkemmin jaksossa 4 xxx, "Kaupallishallinnollisten sovellusten tuottamisen ja ylläpidon automatisointikeinoja".



Kuva 2–18. Automatisoitavissa oleva koodintuotto.

Jaksossa 5, "Tietohakemiston hyväksikäyttö TIP-työkalulla" esittelen näille periaatteille pohjautuvan lisäykseni ohjelmointivälineiden kirjoon, TIP-työkalun. Se on hieman vaikeasti luokiteltavissa mihinkään ohjelmointikielen sukupolveen, vaikka sen perustana onkin tekstitiedostojen "kääntäjämäinen" käsittely. Kuten tullaan näkemään, se on lähempänä tekstintoimitinta kuin varsinaista kääntäjää kielineen — erityisesti, koska TIP:n käsittelemä kieli on rakenteeltaan erittäin yksinkertainen (ks. liite 2). Tekstintoimittimena se on kuitenkin tavanomaisia välineitä melko tavalla vähemmällä käsityöllä toimeen tuleva ja näin se mahdollistaa perinteisten sovelluskehitysvälineiden automaattisemman käytön.

2.4 Jakson yhteenveto

Tässä jaksossa olen pyrkinyt

- näyttämään, kuinka käsitemalli pystyy tiiviissä ja sujuvasti käsiteltävässä muodossa kuvaamaan merkittävän osan tyypillisestä kaupallishallinnollisesta sovelluksesta
- korostamaan, kuinka tietohakemiston käytön hyödyllisyyden ratkaisee pääasiassa sen käyttökelpoisuus kaikkien käytettyjen sovelluskehitysvälineiden kanssa siten, että tietoa monistuu mahdollisimman vähän
- etsimään ratkaisua jatkuvan korjailutyön vähentämiseen siitä, että tavanomaisten sovelluskehitysvälineiden suoran käyttämisen sijaan pyritään tallettamaan asiat, jotka aiottiin tehdä ja varsinainen perusvälineiden käyttö koneellistetaan.

3 TIETOHAKEMISTON RAKENTEESTA

Tässä jaksossa kuvaillaan joustavalle, usean sovelluskehitysvälineen väliselle tietohakemistolle asetettavia vaatimuksia ja tarkastellaan olemassaolevien tietohakemistototeutusten piirteitä. Näiden hyviä puolia kootaan ja puutteita korjataan esimerkkitoteutukseen, jonka prototyyppi toteutettiin tämän työn osana.

3.1 Vaatimuksia ja tavoitteita

Keskeisinä vaatimuksina käyttökelpoiselle tietohakemistolle voidaan pitää kykyä kuvata nykyisen tietotyyppikäsityksen mukaista pysyvää tietoa, kykyä tehdä tämä siten, että tietoa monistuu tietohakemistossa mahdollisimman vähän, ja kykyä sietää muutoksia.

Kyvyllä kuvata nykyistä pysyvän tiedon tietotyyppikäsitystä tarkoitan sitä, että tietohakemistoon on mahdollista tallettaa sellaisia tietotyyppien kuvauksia, jotka ovat edes jossain määrin sopusoinnussa nykyään käytettävien ohjelmointikielten ja tiedonhallintajärjestelmien tyyppijärjestelmien kanssa. Näiden tyyppijärjestelmien piirteisiin kuuluvat mm. sisäkkäiset tietuetyypit, muunnelmatietueet ja nykyään yhä useammin tyyppien välinen ominaisuuksien perintä. Mahdollisuuksien mukaan tietohakemiston pitäisi myös auttaa sellaisen tilanteen hallinnassa, jossa jokin tällainen piirre puuttuu käytettävästä kohdeympäristöstä, esimerkiksi relaatiotiedonhallintajärjestelmästä. Tietohakemiston tyyppijärjestelmää käsitellään lisää kohdassa 3.5.1, "Tietohakemisto ja relaatiotietokannat".

Eräs tapa tarkistaa tietohakemiston kuvauskyky on yrittää kuvata tietohakemiston rakenne sillä itsellään. Tyypillinen tietohakemisto on rakenteeltaan ja tietotyyppikuvauksiltaan melko monimutkainen pysyvä tietojoukko, ja ellei jokin tietohakemistototeutus selviä itsensä kuvaamisesta, on todennäköistä, että monet muutkin monimutkaiset mallinnustilanteet tuottavat sille vaikeuksia.

Tiedon monistuminen tietohakemiston sisällä voidaan minimoida maksimoimalla viittaukset jo kirjattuun tietoon. Tällainen menettely tietenkin monimutkaistaa tietohakemiston ylläpito-ohjelmistoa, koska sen on käyttömukavuuden kannalta tarjottava jonkin verran redundantimpi näkymä ylläpidettäviin tietoihin. Samoin se monimutkaistaa tietohakemiston tietojen hyväksikäyttöä, sillä tarvitut tiedot on koottava useasta lähteestä, joihin ne on jouduttu hajottamaan ei-toivotun toiston välttämiseksi. Lisää viittausten maksimoinnin vaikutuksista on kohdassa 3.5.1, "Tietohakemisto ja relaatiotietokannat".

Muutoksia kestävä tietohakemisto sietää muuttuvia toteutusympäristöjä, useita versioita, asiakaskohtaisuuksia, selväkielisiä kuvauksia usealla kielellä jne. Se pystyy siis kuvaamaan sovelluksen samalla ajanhetkellä käytössä olevia muunnelmia ja myös sovelluksen muuntumista ajan kuluessa. Toisaalta muutoksia sietävää tietohakemistoa itseäänkin voidaan tarvittaessa muuttaa: sen tietosisältöjä voidaan tarkistaa ja jo sen perusrakenne sallii jonkin verran ennalta määräämättömän tiedon tallettamista.

Pelkät joustavat tietosisällöt tietohakemistossa eivät yksin takaa sitä, että kohdassa 2.2, "Tietohakemisto", esitelty tietohakemiston toimintaperiaate toimia erilaisten sovellusvälineiden vaatimien tiedon kuvausten keskitettynä varastona voi toteutua. Näin käy vain, mikäli tietohakemiston hyväksikäyttövälineistö on tarpeeksi joustava. Tämä joustavuus toteutuu, jos nämä välineet asettavat mahdollisimman vähän etukäteisrajoituksia käsiteltävissä oleville ja tuotettaville sovellusvälineiden kuvauksille. Hyväksikäytön menetelmiä käsitellään tarkemmin jaksossa 5, "Tietohakemiston hyväksikäyttö TIP-työkalun avulla".

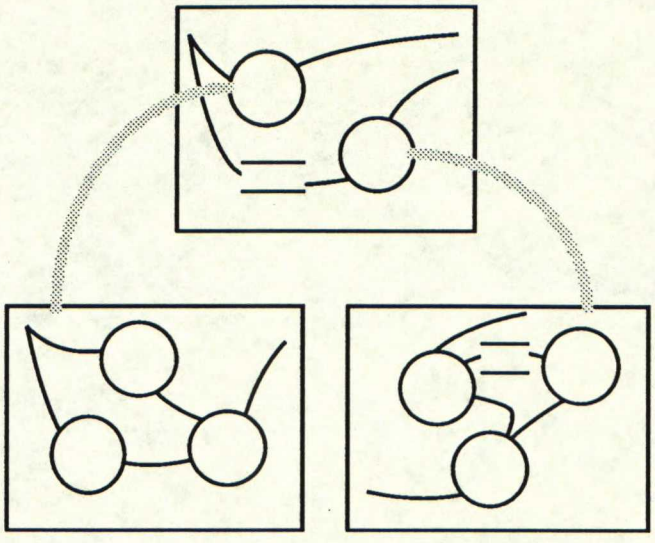
On huomattava, että tietohakemiston loogisen rakenteen kannalta ei ole oleellista, miten tietohakemiston tiedot fyysisesti talletetaan. Fyysisen talletustavan voi tosin nähdä vaikuttaneen monien toteutettujen tietohakemistojen loogiseen rakenteeseen ja tämän takia sitä on käsitelty seuraavan kohdan kuvauksissa. Tavallisia talletustapoja ovat esimerkiksi tekstitiedostot, tarkoitusta varten suunnitellut indeksoidut tiedostot tai varsinaiseen (usein relationaaliseen) tietokantaan talletetut taulut.

3.2 Olemassaolevia tietohakemistototeutuksia

Tutustutaan lyhyesti muutamaan käytännössä olemassaolevaan tietohakemistototeutukseen ja etsitään niistä rakenteellisia tunnuspiirteitä ja tutkitaan niiden mallinnuskykyä ja käyttökelpoisuutta. Erityisesti keskitytään järjestelmässä tarjolla oleviin mahdollisuuksiin hyväksikäyttää tietohakemistoon talletettuja tietoja.

3.2.1 "Structured Analysis/Structured Design"-menetelmän tietohakemistot

SA/SD-menetelmät perustuvat sovelluksen haluttujen toimintojen mallittamiseen ns. tietovirtakaavioiden (engl. *data flow diagram, DFD)* avulla. Tietojärjestelmät nähdään laitteina, jotka muuntavat sisään saapuvat tietovirrat järjestelmästä ulos virtaaviksi tietovirroiksi. Tietojen kulkua kuvaavat tietovirtakaaviot muodostavat hierarkkisesti tarkentuvan kokonaisuuden, jota havainnollistetaan kuvassa 3–1.



Kuva 3–1. Tietovirtakaaviot SA/SD-menetelmässä.

Kaavioiden ympyräsymbolit kuvaavat ns. transformaatioita eli muunnoksia, jotka kuluttavat sisään virtaavia tietoja ja muokkaavat niistä ulos virtaavia tietoja. Mikäli transformaatiot eivät kykene kaikissa tilanteissa jatkuvasti ja välittömästi vastaanottamaan sisäänvirtaavia tietoja, tarvitaan tällaisille tiedoille säilityspaikka, tietovarasto. Näitä kaavioissa merkitään kaksoisviivasymbolilla ja ne ovat sikäli luonteeltaan samankaltaisia kuin tietovirrat, että sekä virran että varaston tietosisältö kuvataan samanlaisin keinoin.[9]

Virroissa kulkevat ja varastoissa säilyvät tiedot kuvataan menetelmässä tietohakemistoon, joka alunperin [DeMarco 1979] ehdotettiin koottavaksi esimerkiksi käsin ylläpidettävään kortistoon. Nykyisissä menetelmän tietokoneavusteisissa työkaluissa on luonnollisesti apuvälineitä tämän kortiston automaattisempaan ylläpitoon.

Menetelmässä jokaisella tietohakemiston alkiolla on (koko kaaviostonlaajuisesti) yksikäsitteinen nimi. Tällainen nimetty tietoalkio voi olla kuvaukseltaan joko alkeistietoa tai se voi koostua muista hakemiston alkioista. Kummankintyyppiseen tiedon kuvaukseen kuuluu olennaisena osana tekstimuotoinen selitys tiedosta. Tämä kommenttiluonteinen osa on ollut selvästi ainakin DeMarcon painopisteenä, sillä määrämuotoisempi kuvaus on tavallaan upotettu tähän tekstiesitykseen antamalla syntaktisia sääntöjä alkeistiedon ja rakenteisen tiedon kuvaamisesta.

[9]Itse asiassa tiettyyn tietovarastoon tulevien ja siitä lähtevien tietovirtojen kuvausten tulisi olla samoja.

Esimerkiksi seuraavanlaisia sääntöjä käytetään antamassa määrämuotoisuutta tietohakemistossa olevalle tekstimuotoiselle kuvaukselle (taulukko 3–1): [Prosa 1989]

Merkintä	Tarkoitus
=	koostuu
... + ...	nämä vaihtoehdot yhdessä
[...|...]	yksi näistä vaihtoehdoista
min{...}max	toistuva osa (minimi-/maksimitoistokerrat)
(...)	valinnainen osa
...	komenttijakso
"..."	merkkijonovakio
'.'	merkkivakio
@	avainkenttä

Taulukko 3–1. SA/SD-tietohakemistomerkinnät.

Huomiota herättävää on, että yksittäisen tietoelementin ominaisuuksien kuten tietotyypin, rajoitteiden jne. määrittely tehdään vapaamuotoisesti kommentteihin, kuten esimerkiksi kuvassa 3–2. [Turtiainen 1990] Tällainen kommentteihin säilöminen vaikeuttaa huomattavasti ko. tietojen automaattista hyväksikäyttöä esimerkiksi erilaisissa sovelluskehitysvälineissä.

```
nopeus =    * tyyppi:real, rajat 0.0 – 100.0, tarkkuus 0.01,
            yksikkö km/h *
```

Kuva 3–2. SA/SD-tietoelementin ominaisuusesimerkki.

Pieni esimerkki tiedon rakenteen määrittelystä SA/SD-tyyppisessä tekstipohjaisessa tietohakemistossa (kuva 3–3): [Turtiainen 1990]

```
nimi          = (titteli) + etunimi + 0{toinen_nimi}2 + sukunimi
titteli       = ["Professori" | "Insinööri" | "Herra" | "Rouva"]
etunimi       = 1{kirjain}n
toinen_nimi   = 1{kirjain}n
sukunimi      = 1{kirjain}n
kirjain       = ['A'-'Ö' | 'a'-'ö']
```

Kuva 3–3. SA/SD-tietohakemistoesimerkki.

Erityisen merkillepantavaa SA/SD-tietohakemistojen tietotyyppimaailmassa on tietotyyppien ja niiden esiintymien (muuttujien) välisen eron hämärtyminen: toisaalta mallin tietovarastot ja -virrat kuvaavat konkreettista tilavarausta muistista tai tiedostoista, toisaalta matalan tason tietoelementtejä käytetään uudelleen

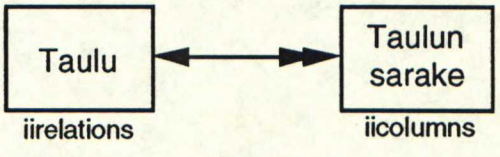
tietotyypin tapaan, kuten vaikkapa kuvan 3–3 esimerkissä tietohakemistovientiä[10] "kirjain".

Useissa menetelmän myöhemmissä muunnelmissa (esimerkiksi [Yourdon 1989]) SA/SD:tä on täydennetty käsite- ja tilakaavioilla. Tietohakemiston kannalta käsitekaavioiden yksilötyypit ovat kolmas tietohakemistoon kuvattava kohde tietovirtojen ja -varastojen ohella.

3.2.2 Kaupallisten relaatiotietokantojen tietohakemistot

Esimerkkinä kaupallisesti saatavilla olevan relaatiotietokannan tietohakemistosta käytän Ingres-järjestelmän [Ingres 1990] järjestelmätauluja (engl. *system tables).* Vastaavia tauluja on käytännöllisesti katsoen kaikissa markkinoilla olevissa relaatiomallin toteutuksissa, sillä ajatus tallettaa relaatiotaulujen rakenne relaatiotauluihin periytyy ensimmäisistä relationaalisista koejärjestelmistä kuten esimerkiksi IBM:n System R:stä.

Kun Ingres:ssä luodaan uusia relaatiotauluja SQL:n "CREATE TABLE"-lausein, taulutiedostojen perustamisen ohella Ingres taltioi taulujen sarakkeiden kuvauksen järjestelmätauluihinsa. Näistä tärkeimmät ovat taulukohtaiset tiedot talletava "iirelations" ja sarakekohtaiset tiedot tallettava "iicolumns".



Kuva 3–4. Ingres-tietohakemiston taulujen kuvaukset.

Ingres käyttää näin säilöttyjä taulujen kuvauksia sisäisesti mm. tulkitessaan SQL-kyselyitä. Sovelluksen tekijälle on julkistettu tietyt dokumentoidut näkymät (engl. *view)* järjestelmätauluihin, joiden luvataan pysyvän muuttumattomina Ingres:n versiosta toiseen ja joiden avulla voitaisiin rakentaa apuvälineitä, jotka perustavat toimintansa sillä hetkellä voimassa oleviin taulujen kuvauksiin. Sen sijaan ainoa tapa muuttaa Ingres-tietohakemiston sisältöä on suorittaa sellaisia SQL:n DDL-lauseita, jotka muuttavat taulurakennetta.

SQL:n tiedon määrittelykieliosuudessa on tosin se ongelma sovelluskehittäjän kannalta, että standardi-SQL:ään ei tarvitse toteuttaa taulujen sarakerakenteen muutoslausetta ("ALTER TABLE"). Tätä ei ole toteutettu myöskään Ingres:ssä, joten ainoa tapa muuttaa jo olemassaolevan taulun sarakkeita on tallettaa taulussa

[10] Käytän sanaa "vienti" englantilaisen "entry"-sanan vastineena.

olevat tiedot väliaikaisesti turvaan, tuhota vanha taulu, luoda se uudelleen uusin sarakkein ja lopulta ladata tähän talletettu tietosisältö. Tämä ei selvästikään ole joustava tapa kehittää eteenpäin käsitemallin yksilötyyppien kuvaamien tietokantataulujen tietosisältöjä, joten Ingres:n tietohakemistosta ei ole suunnitteluvaiheen tietohakemistoksi.

Ingres:iin voi hankkia pelkän tiedonhallintaosan täydennykseksi joukon työkaluja sovellustuotannon avuksi. Näistä tärkeimmät ovat "Vifred", merkkipohjaisten lomakemallien laadintaan soveltuva käyttöliittymäohjelmisto sekä Vifred:n varaan rakentuvat ABF (Applications by Forms) ja QBF (Query by Forms). Kaikki kolme tallettavat niillä rakennetut kuvaukset (lomakemallit, sovelluksen lomakerakenteen, kyselyn tauluviittaukset) järjestelmätauluihin, jotka liittyvät osana Ingres:n tietohakemistoon. ABF-kehittimellä tehdyn sovelluksen lomakkeensisäiset toiminnot laaditaan tavanomaisiin tekstitiedostoihin Ingres:n omalla 4GL-kielellä, ja ainoastaan lomakkeiden välinen sovelluksen (valikko-)rakenne talletetaan tietohakemistoon.

Ingres:llä luotujen sovellusten ylläpitoa helpottava mahdollisuus hyväksikäyttää tiedon (taulujen) rakenteen kuvausta hyväksi Vifred:n, ABF:n ja QBF:n kuvauksissa on tehokkaasti kumottu kahdella tavalla. Koska lähtökohtana on pidetty sitä, että taulujen rakennetta ei voi muuttaa, tätä rakennetta ja yksittäisen sarakkeen kuvauksia monistetaan melko surutta läpi em. välineiden. Toisena hyväksikäytön vaikeutena on se, että ABF:n 4GL-kieli sisältää yllättävän vähän rakenteita, joilla pyydettäisiin järjestelmää toteuttamaan jotain taulujen tietosisällöstä riippuvaa. Esimerkkinä tällaisesta puuttuvasta konstruktiosta voisi olla vaikkapa "siirrä SQL-kyselyn lopputulosriviltä kaikki samannimiset kentät lomakkeelle". Jotkin kehittimen osat sentään hakevat oletusarvomaisia pohjatietoja: esimerkiksi uutta lomaketta luotaessa voidaan pyytää yhden taulun kentät lomakkeelle pohjaksi riveittäisessä tai sarakkeittaisessa oletussijoittelussa. Tästä toimenpiteestä ei taltioidu kuitenkaan mitään pysyvää yhteyttä luodun lomakkeen ja vastaavan tietokannan taulun välille — myöskään sellaista päivitystoimenpidettä ei ole saatavilla, jolla voitaisiin "virkistää" lomakkeen ja tietokantataulun välillä samannimisten kenttien tuoreimmat tietokuvaukset lomakkeelle.

3.2.3 Kaupallisten CASE-välineiden tietohakemistot

Nykyiset kaupalliset tietokoneavusteisen systeemityön välineet perustuvat hyvin usein SA/SD-menetelmään. Välineiden tämänhetkinen kattavuus ei ole erityisen laaja: niiden ainoana tavoitteena näyttää olevan auttaa tyylikkäiden SA/SD-suunnitelmia luomisessa ja tässä ne onnistuvatkin melko hyvin.[11] Sen sijaan lähes

poikkeuksetta välineillä luotujen suunnitelmien jatkohyväksikäyttö on jätetty vähemmälle huomiolle tai esimerkiksi kolmansien osapuolien tehtäväksi. Sovelluksen toteutuksen integrointi suunnitteluapuvälineeseenon löyhä, erityisesti silloin, kun suunnitelman muutosten pohjalta pitäisi tehdä sovellusmuutoksia.

Koska nämä välineet pohjaavat SA/SD-menetelmään, ne sisältävät erilaisia toteutuksia menetelmässä "klassisesti" käytetystä tietohakemistosta, jollaista kuvattiin kohdassa 3.2.1, "Structured Analysis/Structured Design -menetelmän tietohakemistot". Seuraavassa käsitellään muutaman välineen tietohakemistototeutusta Tietosavo Oy:ssä vuonna 1991 toteutetussa välinetutkimuksessa saatujen kokemusten perusteella.

3.2.3.1 Insoft Prosa, Cadre Teamwork

Näillä kahdella välineellä on keskenään niin paljon yhteisiä piirteitä, että päällekkäisyyksien välttämiseksi käsittelen ne samassa kohdassa.

Sekä oululaisen Insoft Ky:n "prosa" [Prosa 1989] että amerikkalaisen Cadre Technologies Inc:in "Team*work*"[12] [Cadre 1988] ovat SA/SD-menetelmän tietokoneavusteisia apuvälineitä. Molemmat toimivat usean valmistajan Unix-työasemissa, Prosa myös DOS- ja OS/2-mikrotietokoneissa, ja ne tukevat sellaista nykyään melko usein käytettyä SA/SD-menetelmän laajennusta, jossa menetelmään mukaan on otettu tietovirtakaavioiden (DFD) lisäksi myös käsite- (ERD) ja tilakaaviot (engl. *state transition diagram, STD).* Prosa tallettaa piirretyt kaaviot kunkin omaan tekstitiedostoonsa, Teamwork käyttää puolestaan kuvaustietokannaksi kutsumaansa ratkaisua, jossa työasemaverkon yhdessä koneessa toimiva palvelinprosessi huolehtii suuresta joukosta hierarkkiseen tiedostojärjestelmään sijoiteltuja tiedostoja.

Molempien välineiden käsitys tietohakemiston sisällöstä on samanlainen: tietovirtakaavion tietovirtaan ja tietovarastoon sekä käsitekaavion yksilötyyppiin voidaan liittää monirivinen tekstimuotoinen kuvaus. Kumpikaan väline ei aseta erityisiä rajoituksia esimerkiksi talletettavan tekstin pituudelle tai kuvauksen sisällölle. Molemmissa välineissä tietohakemiston raportointityökalut odottavat kuvauksilta kuitenkin kohdassa 3.2.1, "Structured Analysis/Structured Design -menetelmän tietohakemistot"" esiteltyjä syntaktisia sääntöjä.

---

[11] Joidenkin välineiden kyky käsitellä todellisen kokoisia suunnitelmia jää kokeilujen perusteella tosin arvoitukseksi: suorituskyvystä voisi kuitenkin ekstrapoloida, että jotkin välineet on suunniteltu lähinnä SA/SD-oppikirjojen havaintoesimerkkien toistamiseen.

[12] Kirjoitusasut yritysten tuotteistaan käyttämiä. Lopussa tekstissä käytän nimiä "Prosa" ja "Teamwork" ilman erityiskorostuksia.

Välineet eroavat tavassa, jolla edellä kuvatut "tekstilaput" talletetaan. Prosassa tekstirivit talletetaan sen kaaviotiedoston sisään, jossa määrittelyn kohteena oleva symboli on. Teamwork:ssa tietohakemistoviennit talletetaan yhteen suureen kokoelmaan, jossa kukin vienti on nimetty (pitkähköllä) kaaviostonlaajuisesti yksikäsitteisellä nimellä. Tämä nimi talletetaan niihin kaaviosymboleihin (ja toisiin tietohakemistovienteihin), joista halutaan viitata ko. vientiin. Prosassa tällainen ristikkäisviittaus järjestetään tallettamalla viittaavaan paikkaan kohdekaaviotiedoston nimi. Prosa rakentaa tällaisia viittauksia myös automaattisesti silloin, kun tietovirtakaavion ylätasolta tarkennetaan hierarkiassa tarkemmalle tasolle. Tällöin kaikki ylemmältä tasolta alemmalle siirtyvät tietovirrat saavat automaattisesti tietohakemistokuvauksekseen viittauksen ylemmässä kaaviotiedostossa sijaitsevaan kuvaukseen. "Vieraassa" kaaviotiedostossa sijaitsevia tietohakemistovientejä voidaan käsitellä lähes yhtä helposti kuin kaavion "omia" vientejä, joten välineen käyttäjän kannalta lopputulos on molemmissa tuotteissa lähes sama.

Teamwork:ssa voi luotaville uusille tietohakemistovienneille määritellä oletuspohjan, jonka teksti kopioidaan uuden viennin lähtökohdaksi. Tämän pohjan avulla voidaan ehkä saavuttaa jonkintasoista määrämuotoisuutta kuvausten kesken. Tietohakemistoviennin muuttamisen jälkeen Teamwork tallettaa siitä kokonaan uuden version ja jättää vanhan version vielä talteen mahdollista myöhempää (käsin tai koneellisesti tapahtuvaa) käsittelyä varten.

Molempien välineiden pääasiallinen tapa käyttää tietohakemistoa hyväksi on raportoida sen sisältöä erilaisiin tarkoituksiin — aakkosjärjestyksessä oleva luettelo määrityksistä, missä eri vientejä on käytetty, kieliopiltaan virheelliset määritykset jne. Prosan raportointiväline on melko laajasti konfiguroitavissa (hakuehdot, tulostusjärjestys, tulosteen ulkoasu), mutta toisaalta Teamwork:in valmiiden raporttien joukossa on mm. tietohakemistoon kuvattujen sisäkkäisten tietorakenteiden koko rakenteen avaava raportti.

Sovellusta päästään toteuttamaan laadittujen kaavioiden pohjalta kummankin välineen tapauksessa vasta erillistuotteiden lisähankinnan myötä. Prosaan on saatavissa piirrettyjen käsitekaavioiden yksilötyyppien tietohakemistovienneistä SQL-kielisiä taulujenluontilauseita tuottava "prosaSQL"-tuote. Insoft Ky:llä on tiettävästi nykyään tarjota myös tietyllä tavalla piirretyistä tietovirta- ja tilakaavioista C-kielistä ohjelmakoodirunkoa tuottava erillistuote. Tämän tuotteen yhteydessä tietovirtojen tietohakemistomäärityksistä tuotetaan C-kielisten funktioiden välistä parametrinvälitystä ja tietovarastoista vastaavasti muuttujamäärityksiä.

Vastaavia erillistuotteita on saatavilla myös Teamwork:n yhteyteen kolmansilta

osapuolilta. Eräs niistä kykenee suorittamaan/simuloimaan tietyllä tavalla loppuunsaatettua tietovirta- ja tilakaavioiden yhdistelmää varustettuna välineellä laaditulla graafisella käyttöliittymällä prototypointitarkoituksia varten. Toinen tuote pystyy ProsaSQL:n tapaan tuottamaan käsitekaavioiden yksilötyyppien tietohakemistokuvauksista taulujen luonti-SQL-lauseet, mutta lisäksi se pystyy tuottamaan sellaiset SQL-lauseet, joiden avulla vanhoista tietokantatauluista saadaan siirrettyä sisältö turvaan uusiin tauluihin. Tämän toteuttamiseen väline käyttää Teamwork:in kykyä tallettaa muutostilanteissa aiemmat versiot tietohakemistovienneistä. Teamwork:ssa on lisäksi ohjelmointiliittymä, jonka avulla kuvaustietokantaan talletettuja asioita (ml. tietohakemistoviennit) voidaan lukea ja muuttaa itse tehdyin apuohjelmin. Tämän vastineeksi Prosan käyttämä kaaviotiedostojen formaatti on dokumentoitu: se on muodoltaan Prolog-pohjainen tekstitiedosto, joka sisältää kaavioiden sisältämät tosiasiat (faktat) muodossa, jota voidaan kohtuullisen helposti jatkokäsitellä omin ohjelmin. Kaavioiden graafista osaa ylläpidetään toisessa tekstitiedostossa ja tätä tiedostomuotoa ei ole dokumentoitu, joten kaavioiden muuttaminen omin apuohjelmin vaatii hieman kekseliäisyyttä ja kokeilua, mutta on käytetyn tekstipohjaisen rakenteen ansiosta ainakin periaatteessa mahdollista.

3.2.3.2 Intersolv Excelerator

Amerikkalaisen Intersolv Corp:n (aiemmin Index Technology Corp.) DOS-ympäristöön tuottama "Excelerator/IS" on nimenomaan kaupallishallinnollisten järjestelmien SA/SD-suunnittelun apuväline. [Excelerator 1989] Tietovirta-, käsite- ja tilakaavioiden ohella järjestelmällä voidaan suunnitella mm. merkkipohjaisia näyttöjä ja tulostemalleja. Järjestelmä on hieman iäkkäämpi kuin esimerkiksi Prosa ja tämä näkyy jonkin verran mm. järjestelmän osien pilkkoutumisena lukuisiin (kooltaan riittävän pieniin) DOS-ohjelmiin, joita kehysjärjestelmä käynnistää vuoronperään tarvittavan toiminnon mukaan. Myös järjestelmän kaavioiden piirtoon käytettävä osa on kankeahko käyttää.

Excelerator:n tietohakemisto poikkeaa Prosan ja Teamwork:n vastaavista toteutukseltaan, mikä on vuorostaan vaikuttanut tietohakemiston rakenteeseen. Intersolv nimittäin kutsuu tietohakemistoksi sitä suurikokoista indeksoitua tiedostoa, johon kaikki Excelerator:lla tehtyyn suunnitelmaan kuuluva tieto talletetaan. Tämän tiedoston tärkeimpänä roolina on toimia kaiken muun kuin varsinaisten tietohakemistovientien talletuspaikkana: esimerkiksi tietovirta- ja käsitekaavioissa esiintyvät graafiset elementit talletetaan myös tähän samaan keskitettyyn paikkaan. Mainitun indeksoidun tiedoston rakenne on erittäin suoraviivainen: siinä on 6 kilotavua pitkiä tietueita, joiden avaimena on tietueen kolmimerkkisen tyy-

pin ja 32-merkkisen nimen yhdistelmä. Teknisesti toteutuksesta on saatu varsin joustava menettelyllä, jossa useiden tietuetyyppien käsittely hoidetaan samalla ohjelmalla, joka vain parametroidaan kulloinkin tarvittavalla kenttäkuvauksella: minkänimisiä kenttiä sijoitetaan mihinkin tietueen tavupositioon. Niinpä tietohakemiston tietosisältöjen kehittäminen on tuon 6 kilotavun puitteissa melko helppoa ja Intersolv onkin käyttänyt tätä hyväkseen tuomalla tuotteestaan markkinoille myös "Excelerator/RT"-version, jossa joitain yksityiskohtia on muokattu sopivammaksi tosiaikajärjestelmien kuvaamiseen kajoamatta kuitenkaan varsinaisiin suoritettaviin ohjelmiin.

Tästä kiinteänmittaisen tietueen uudelleenkäytöstä seuraa se Excelerator:n piirre, että tuote ei salli minkäänlaista vapaamuotoisuutta tietohakemistovienneissä (tai muissa talletettavissa tiedoissa). Tietohakemistoon voi tallettaa vientinä yksittäisen tietoelementin kuvauksen tai vaihtehtoisesti rakenteisen tiedon kenttäluettelon periaatteeltaan samaan tapaan, mikä esitettiin kohdassa 2.2, "Tietohakemisto" kuvissa 2–7 ja 2–8. Jokaisesta tietoelementistä talletetaan ennalta määrätyt tiedot (pari kuvaruudullista). Rakenteisten tietojen kuvaus on yksinkertaisesti luettelo tietoon kuuluvista toisista tiedoista: jokaisella tietueen kentällä täytyy siis olla tuo pari kuvaruudullista tietoelementtimääritystä. Kaikilla tietohakemistovienneillä on kaaviostonlaajuisesti yksikäsitteinen nimi, mikä sointuukin hyvin käytettyyn tiedostototeutukseen.

Excelerator käyttää tietoelementtien tyypin ja muotoilun esittämiseen COBOL-kielen tietomäärittelyn syntaksia eli ns. "PICTURE"-kuvauksia. Tämä on ollut käytännöllistä kahdesta syystä: "PICTURE"-kuvaus kykenee esittämään sekä tiedon tilanvarauksen että sen muotoilun samalla kertaa ja se on yksinkertaisimmin käytettävissä niissä kolmansilta osapuolilta saatavissa tuotteissa, jotka tuottavat COBOL-kielisiä ohjelmarunkoja Excelerator:sta tekstimuotoon siirretyistä kaavio- ja tietohakemistotiedostosta. Toisaalta näin sovelluskehitysvälinekohtaisen kuvaustavan käyttö vaikeuttaa selvästi jonkin muun ohjelmointikielen käyttöä Excelerator:lla suunnitellun sovelluksen toteutukseen.

Sen periaatteen ansiosta, että kaikkia tietoja voidaan hakea samanlaisella menettelyllä, viittauksia talletettujen tietojen välillä on käytössä hyvin runsaasti — lähes kaikkiin asioihin kuten kaavioihin, niiden symboleihin, tietohakemistovienteihin jne. voidaan liittää huomautuksia, ko. asian asettamia ja täyttämiä vaatimuksia, mitä tahansa toisia kaavioita jne. Tätä vasten tarkasteltuna onkin yllättävää, että varsinaisen tietohakemiston sisällä on tyydytty vain kaikkein yksinkertaisimman viittauksen toteuttamiseen: viittaus rakenteisen tiedon kuvauksesta toisiin tietokuvauksiin. Erityisesti se, ettei tietoelementtien kuvauksia voi kerrostaa tavalla, joka esiteltiin kohdassa 2.2, "Tietohakemisto" kuvassa 2–14, aiheuttaa lisätyötä ja ylläpito-ongelmia keskenään vain hiukan eroavien tietoelementtien kuvausten

ollessa kokonaan erilliset. Tämä tietotyyppien puuttuminen on toki täysin sopusoinnussa COBOL-kielen tietokäsityksen kanssa, jossa sekä tietoelementit että niistä koostetut tietueet ovat olemassa ikäänkuin vain kerran.

Myös Excelerator:n tietohakemiston pääkäyttötarkoitus on Prosan ja Teamwork:n tapaan tuottaa erilaisia raportteja tietohakemiston sisällöstä. Merkkipohjaisia näyttö- ja tulostemalleja laadittaessa on mahdollista noutaa malliin tulevan kentän "PICTURE"-kuvaus tietohakemistosta, mutta tätä yhteyttä ei muodosteta pysyväksi siten, että kentän kuvauksen muuttuessa tietohakemistossa myös kaikki näyttö- ja tulostemallit, joissa kenttä näkyy, päivittyisivät automaattisesti. Raportoinnilla voidaan sentään etsiä kaikki näytöt ja tulosteet, joissa muuttunut kenttä esiintyy ja korjata kenttien ulkoasut käsin. Sekä käsitemallin yksilötyyppien tietohakemistovienneistä että näyttö- ja tulostemalleista voidaan tuottaa raportin tapaisella menettelyllä COBOL-kielisiä tietuekuvauksia.

Myös Excelerator:iin on jätetty mahdollisuus tehdä omia työkaluja, jotka manipuloivat sillä rakennettuja suunnitelmia mukaanlukien suunnitelman sisältämä tietohakemisto. Kuvauksen (em. keskitetyn tiedoston) voi siirtää dokumentoituun tekstimuotoon, muokata tätä ja siirtää takaisin Excelerator:n sisäiseen muotoon. Lievänä ongelmana tällaisten apuvälineiden teossa on se, että kuvauksen tietosisällön sanelevat Excelerator:ssa käytetyt tietosisällöt, jotka saattavat vielä muuttua tuotteen kehittyessä myöhemmissä versioissa. Tietosisältöjen muuttaminen on tosin mahdollista erillisenä hankittavissa olevalla "Customizer"-tuotteella.

Oman ongelmansa Excelerator:n käytössä todellisessa sovellustuotannossa muodostaa sen DOS-pohjaisuus yhdessä yhteen suureen tiedostoon keskitetyn kuvausten taltiointitavan kanssa. Tuote ei nimittäin tue ko. kuvaustiedoston yhteiskäyttöä usealta työasemalta käsin. Tähän korjauksen pitäisi tuoda (tuotteeseen tutustumishetkellä 1991) luvassa ollut ja nykyään tiettävästi saatavilla oleva OS/2-versio, jossa kuvausten talletus on toteutettu Teamwork-tyyppisellä verkon yhdessä asemassa toimivalla palvelinprosessilla.

3.2.4 IBM Repository Manager/MVS

1980–1990-lukujen taitteessa IBM toi julkisuuteen mittavan hankkeensa kaupallishallinnollisen sovelluskehityksen puitemalliksi, jota yritys kutsuu nimellä "AD/Cycle". [IBM2 1991] Mallin tarkoituksena on luoda yhteiset toimintaperiaatteet ja puitteet sovellustuotannon kaikissa vaiheissa käytettäville menettelytavoille ja työkaluille. Keskeinen puitemalliin kuuluva työkalu on keskitetty tie-

tohakemisto, "repository", joka toimii yhteisenä tukipisteenä mallissa käytettäville muille sovellusvälineille. Tämän tietohakemiston käytännön toteutus on IBM-suurkoneympäristössä MVS-käyttöjärjestelmän ja DB2-relaatiotiedonhallintajärjestelmän avulla toteutettu tuote, "Repository Manager/MVS". Käytän jatkossa tuotteesta lyhennettä "RM/MVS".

RM/MVS:n tallettamien kuvausten runkona on käsitemallia muistuttava informaatiomalli, vieläpä siten, että järjestelmän toimitukseen sisältyvät valmiit mallit RM/MVS:n tallettamista tiedoista. [IBM3 1990] Näitä kuvauskannan itsensä kuvauksia IBM kutsuu nimellä "AD/Cycle Information Model" ja niiden tarkoituksena on toisaalta tarjota perusvälineet sovellusalueen hakemistoitavan tiedon tallettamiseen ja toisaalta mahdollistaa kuvauskannan muokkaus johonkin tiettyyn käyttötarkoitukseen.

Koska RM/MVS:n keskeisenä tavoitteena on ollut tukea sovellustuotannon eri vaiheisiin liittyviä useita, mahdollisesti eri valmistajilta tulevia välineitä, lähtökohtana on ollut varmistaa kuvauskannan laajentamis- ja muokkauskestävyys. Tähän on päästy toisaalta informaatiomallin kuvauksella, toisaalta kuvauskantaa manipuloivien ns. työkalujen (engl. *tool)* välityksellä. Tärkeämpää kuin toteuttaa pelkkä tietohakemiston ylläpitoon soveltuva ohjelmisto on IBM:n mielestä ollut laatia tarvittavat työkalut uusien työkaluohjelmien laatimista varten. Tietohakemistoa päivittävät työkalut voivat olla hyvin monimuotoisia: esimerkiksi merkkipohjaisia ylläpitolomakkeita, erityistarkoituksiin laadittuja ohjelmia tai OS/2-työasemassa graafisen käyttöliittymän varassa toimivia apuvälineitä. Perustyökalut tulevat luonnollisesti RM/MVS:n toimituksen mukana, ja näiden joukkoon kuuluu sekä merkkipohjaisena 3720-päätteiltä toimivia ylläpitolomakkeita että OS/2-työasemassa toimiva graafinen käsitemallin muokkausohjelmisto "RMGraph".

RM/MVS:n tietohakemiston fyysisenä talletuspaikkana on suuri joukko DB2-tietokannan relaatiotauluja. [IBM4 1991] Taulujen lukumäärän huomioon ottaen (yli 250 kappaletta) on luonnollista, että RM/MVS:iin sisältyy työkalut taulujen automaattiseen luontiin suoraan käsitemalliin sisällytetyistä attribuuttiluetteloista. Tapahtumaa kutsutaan talletusnäkymän sitomiseksi (engl. *binding of a storage view),* jonka RM/MVS tekee automaattisesti useimmissa käsitemallin muutostilanteissa.

RM/MVS:iin kuuluu joukko työkaluja, joiden avulla laaditun käsitemallin käyttäytymistä voidaan tarkastella laatimatta varsinaista testisovellusohjelmaa. Käsitemallin yksilötyypeistä voidaan ilman lisämäärityksiä tallettaa esiintymiä testitarkoituksiin, ja mallista voidaan poimia otteita (engl. *aggregate),* joiden varaan voidaan varsin automaattisesti laatia käyttönäkymiä (engl. *logical view),* joiden

alalajia "varmennusfunktio" (engl. *validation function)* voidaan vuorostaan ajaa kyselemään ja manipuloimaan usean yksilön käsitemallin määräämissä suhteissa olevia kokonaisuuksia. Varmennusfunktio käyttää tietojen esittämiseen joko automaattisesti laadittua lomaketta (engl. *panel)* tai vaihtoehtoisesti lomake voidaan laatia itse.

Vaikka lähes kaikkia RM/MVS:n tietohakemistoon kuuluvia tietoja voidaan käsitellä RM/MVS:n mukana tulevilla merkkipohjaisilla lomakkeilla, varsinainen suunniteltu kuvauskannan pääkäyttötapa on käyttää sovelluskehitysvälineitä, jotka integroituvat kuvauskantaan RM/MVS:n palvelujen avulla muun sovellustyöskentelyn ohessa. Esimerkki tällaisesta tuotteesta on RM/MVS:n mukana tuleva OS/2-työasemassa toimiva käsitekaavioiden laadintaohjelma "RMGraph", joka tallettaa kaavioiden piirron lomassa niiden tietoja RM/MVS:iin. Näin esimerkiksi uuden yksilötyypin lisääminen käsitekaavioon saadaan vietyä toteutukseen asti lähes välittömästi piirroksen tallennuksen yhteydessä, jonka jälkeen uusien yksilöiden käyttäytymistä voidaan testata hyvin vähäisellä lisätyöllä.

Yksilötyypin tietosisältö kuvataan IBM:n toimittamassa pohjamallissa relaatiotietokannoille soveltuvassa rakenteettomassa luettelomuodossa. Mikään ei tietenkään estä lisäämästä malliin sisäkkäisiä tietotyyppejä kuvaavia tietohakemiston kohteita ja kehittämästä sellaista työkaluohjelmaa, jonka avulla sisäkkäisyys puretaan vaikkapa suoraan alkuperäisen mallin yksilötyyppien kenttäluetteloihin.

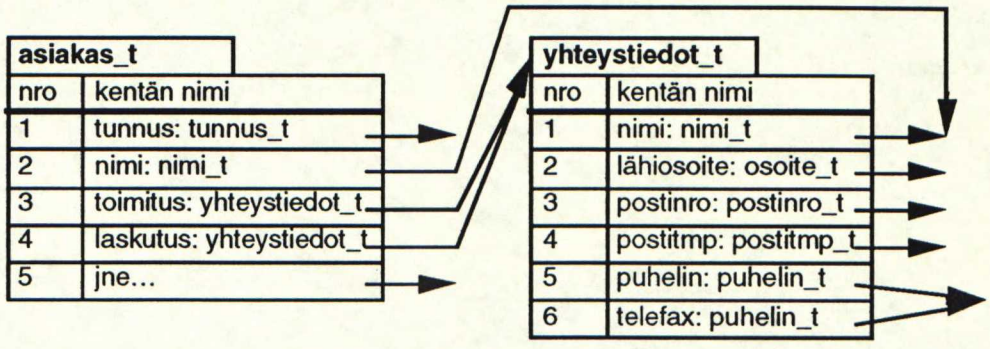
3.2.5 Kaupallisten tietohakemistojen yhteisiä piirteitä

Keskeisenä kaupallisia tietohakemistoja yhdistävänä piirteenä on kyky palvella lähinnä vain sitä välinettä, joka sisältää ko. tietohakemistototeutuksen. Mikäli väline on määrittely- ja suunnittelutyön apuväline, ponnistukset integroida tietohakemistoja ylipäänsä johonkin toteutustason sovelluskehitysvälineeseen ovat vielä melko vaatimattomia. Parhaimmillaankin on mahdollistettu lähinnä tietohakemiston kuvausten yksisuuntainen siirto toteutusvälineisiin — pysyvän, tietohakemistomuutokset sallivan yhteyden muodostuminen on hyvin harvinaista.

Muutokset salliva yhteys tietohakemiston ja vapaasti valittavien perinteisten sovelluskehitysvälineiden välillä on hyväksikäytön harvinaisin muoto. IBM:n RM/MVS vaikuttaa ainoalta tämän työn puitteissa esilletulleelta kaupalliselta tuotteelta, jolla on selviä pyrkimyksiä tällaisen hyväksikäytön suuntaan.

SA/SD-pohjaiset tietohakemistot kärsivät kohdassa 3.2.1, "Structured Analysis/Structured Design -menetelmän tietohakemistot " esiin tulleesta tietotyyppi-

käsityksen hämärtymisestä. Kuvassa 3–5 on havainnollistettu tilannetta, jossa samoja tietotyyppejä ("yhteystiedot_t", "puhelin_t", "nimi_t") halutaan käyttää useaan kertaan saman rakenteisen tiedon kuvauksen sisällä.



Kuva 3–5. Tietotyyppiviittaukset.

Perinteisessä SA/SD-tietohakemistossa joudutaan luomaan ylimääräisiä avustavia tietohakemistovientejä tällaisten viittausten mahdollistamiseksi. Viittaukset sallivan tietotyyppijärjestelmän rakennetta käsitellään lisää kohdassa 3.5.1, "Tietohakemisto ja relaatiotietokannat".

3.3 Esiteltävän tietohakemistototeutuksen lähtökohtia

Seuraavissa kohdissa esittelen viittauksia paremmin hyväksikäyttävän ja useat rinnakkain käytettävät sovellusvälineet huomioonottavan tietohakemiston toteutusperiaatteita.

Prototyyppiä toteutettaessa fyysiseksi talletustavaksi valittiin relaatiotietokanta pääasiallisesti tiedonhallintajärjestelmän peruspalveluiden tarjoamien hyötyjen takia, joita ovat mm. usean yhtäaikaisen käyttäjän tuki, varmistus- ja toipumismenettelyt ja vakiomuotoinen SQL. Usean yhtäaikaisen käyttäjän tuki sallii usean sovellustyöntekijän käsitellä saman sovelluksen kuvauksia. Varmistus- ja toipumismenettelyt parantavat tietohakemiston käyttöluotettavuutta. SQL tietohakemiston kysely- ja manipulointirajapintana on varsin joustava tapa hyväksikäyttää hakemiston tietoja. Relaatiotaulut ovat myös esimerkiksi kiinteitä tiedostoratkaisuja helpompia muokata ja laajentaa.

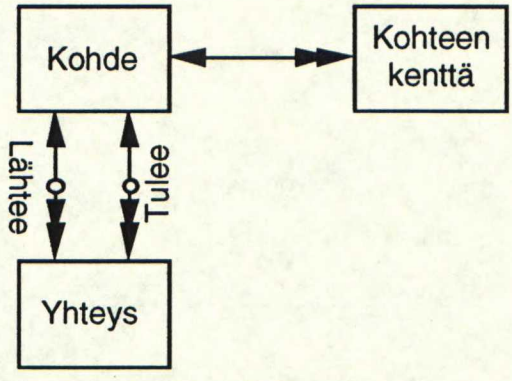
Myös tietohakemistoon kuvattavan sovelluksen toteutuksessa oletettiin käytettävän relaatiotietokantaa. Tämä ei kuitenkaan välttämättä estä muunkintyyppisten tietokantojen käyttöä. Jotkin tietohakemiston tarjoamat palvelut suuntautuvat tosin spesifisesti relaatiotietokantojen puutteiden (sisäkkäisten tietueiden puute, hankalat taulumuutokset) lievittämiseen, mikä saattaa olla hyödytöntä johonkin

toisentyyppiseen tietokantaan sovellettuna. Jonkin verran pyrittiin välttämään esimerkiksi kohdetiedonhallintajärjestelmän käyttämien alkeistason tietotyyppikuvausten säilömistä tietohakemiston sisältöön, ja muitakin samantapaisia tietokantariippuvaisia asioita on eristetty TIP-työkalun avulla. (Lisätietoja jaksossa 5 ja liitteessä 3.)

Tietohakemistototeutus tehtiin perustumaan käsitemalliin siten, että käsitemallin yksilötyypit vastaavat tarkoin sovelluksen tarvitsemia relaatiotauluja. Yksilötyypin tietosisältö kuvataan tietohakemistoon sisältyvällä tyyppijärjestelmällä, joka lisää tiedon rakenteen kuvausten uudelleenkäytettävyyttä. Kohdassa 2.1, "Käsitemalli" esiteltyjä sovelluksen lomakeohjelmia kuvaavia käsitemallin otteita voidaan tallettaa tätä varten laadittuun tietohakemiston osaan.

3.4 Käsitemalli tietohakemistossa

Toteutetun tietohakemiston ytimenä on käsitemallin yksilötyypit ja yhteydet taltioiva taulupari "Kohde" ja "Yhteys", joka on esitetty kuvassa 3–6.



Kuva 3–6. Käsitemalli tietohakemistossa.

Kukin käsitemallin yhteys päätettiin tallettaa kahtena rivinä "Yhteys"-tauluun eli kumpikin yhteyden lukusuunta omana rivinään.[13] Rivi "Yhteys"-taulussa kertoo, kuinka monta "Tulee"-suunnan kohdetta vähintään (0, 1) ja enintään (1, n) on kutakin olemassaolevaa "Lähtee"-suunnan kohdetta kohti. Tällä tavoin talletettuna on helppo löytää kaikki kohteesta "poispäin" etenevät yhteydet riippumatta siitä, kumpaako yhteyden suuntaa on pidetty käsitemallia talletettaessa pääasiallisena suuntana. Haluttaessa on mahdollista tallettaa myös sellaisia yhteysrivejä, jotka kertovat, että yhteyttä näiden kahden kohteen välillä ei tähän suuntaan ole (sekä vähimmäis- että enimmäismäärä on 0), esimerkiksi dokumentoimaan se, että ko. suunta on suunniteltaessa tarkastettu ja tarpeettomaksi ha-

[13]Mikäli yhteyttä toiseen suuntaan ei ole (sekä vähimmäis- että enimmäismäärä on 0), voidaan toinen rivi haluttaessa jättää pois.

vaittu. Yhteys yksilöidään niiden kohteiden tunnuksilla, joiden välissä yhteys on. Mikäli kahden kohteen välillä on useita yhteyksiä, ne erotellaan toisistaan roolitunnuksella. Roolitunnus on samalla ainoa tapa nimetä yhteyksiä.

"Kohde"-tauluun talletettavat käsitemallin yksilötyypit vastaavat suoraan toteutettavan relaatiotietokannan tauluja. Yhteyksistä ei tauluja tarvitse tehdä, sillä "n:m"-tyyppisiä tai yhteyden ja yksilötyypin välisiä yhteyksiä ei sallita talletettavan. Perusteluna tälle rajoitukselle on se, että näiden toteuttaminen vaatii relaatiomallissa joka tapauksessa välittävän taulun. Ensimmäinen sääntö (kohteiden ja toteutettujen taulujen vastaavuus) on haluttu pitää voimassa myös toisinpäin, jolloin jokaista toteutettua taulua kohti pitää olla olemassa vastaava tietohakemiston kohde.

Tauluun "Kohteen kenttä" taltioidaan relaatiomallin mukainen kentittäinen eli "litteä" luettelo kohteen elementtitason kentistä. Tätä luetteloa ei laadita käsin, vaan se kuvataan kohdassa 3.5.1, "Tietohakemisto ja relaatiotietokannat", esitetyllä tyyppijärjestelmällä.

Esimerkkinä näytän, kuinka kohdassa 2.1.2, "Käsitemalli ja kaupallishallinnollisen sovelluksen rakenne" esitetty "Asiakas-tilaukset"-käsitemalli voitaisiin tallettaa kuvan 3–6 käsitemallista johdettuihin relaatiotietokantaan luotuihin tietohakemistotauluihin (taulukot 3–2 ja 3–3).

Kohde			
tunnus	lyhyt_tunnus	nimi	...
Asiakas	asi	Asiakas	
Tilaus	til	Tilaus	
Tilausrivi	tir	Tilausrivi	
Tuote	tuo	Tuote	
Tuotteen_osa	tos	Tuotteen osa	
Tuotetta_varastossa	tva	Tuotetta varastossa	
Varasto	var	Varasto	

Taulukko 3–2. Käsitemallin kohteet tietohakemistossa.

Yhteys					
lähtee	tulee	rooli	min	max	...
Asiakas	Tilaus		0	n	
Tilaus	Asiakas		1	1	
Tilaus	Tilausrivi		1	n	
Tilausrivi	Tilaus		1	1	
Tilausrivi	Tuote		1	1	
Tuote	Tilausrivi		0	n	[1]
Tuote	Tuotteen_osa	Koostuu	0	n	
Tuotteen_osa	Tuote	Koostuu	1	1	
Tuote	Tuotteen_osa	Osana	0	n	
Tuotteen_osa	Tuote	Osana	1	1	
Tuote	Tuotetta_varastossa		0	n	
Tuotetta_varastossa	Tuote		1	1	
Tuotetta_varastossa	Varasto		1	1	
Varasto	Tuotetta_varastossa		0	n	
[1] Yhteyttä tuskin käytetään sovelluksessa.					

Taulukko 3–2. Käsitemallin yhteydet tietohakemistossa.

3.5 Tietohakemisto ja sovellustyövälineet

Tarkastellaan tietohakemiston suhdetta kolmeen kaupallishallinnollisia sovelluksia rakennettaessa tänä päivänä yleisesti käytettyyn sovelluskehitystyövälineen lajiin: relaatiotietokantaan, käyttöliittymäohjelmistoon ja ohjelmointikieleen.

3.5.1 Tietohakemisto ja relaatiotietokannat

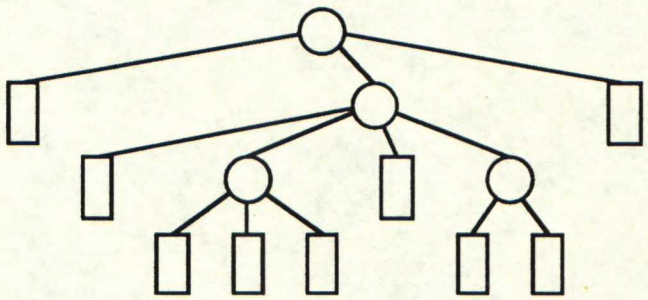
Kuten aiemmin kohdassa 3.2.2, "Kaupallisten relaatiotietokantojen tietohakemistot", todettiin, lähes jokaisessa relaatiotietokannan toteutuksessa on mukana jonkinlainen tietohakemisto. Nämä tietohakemistot ovat tosin yleensä keskittyneet "muistamaan" kerran luotujen taulujen rakenteen (SQL:n "CREATE TABLE"-lauseet) eikä näin talletettuja tietoja yleensä käytetä juuri muuhun.

Relaatiotietokantojen ja 3GL-ohjelmointikielten välinen "impedance mismatch" on tunnettu seikka kirjallisuudessa: relaatiotietokannat palauttavat hakujen tulokset joukkoina rivejä (engl. *set of tuples),* mutta perinteiset ohjelmointikielet käsittelevät tietoja tietue (rivi) kerrallaan. Huomattavasti harvemmin käsitelty aihe on relaatiotietokantojen ja ohjelmointikielten tyyppijärjestelmien välinen "maintainability mismatch": siinä, missä ohjelmointikielten tietotyyppien (engl. *data types)* muutokset on suunniteltu vaikuttamaan automaattisesti (ohjelmointikielen kääntäjän "levittäminä") jokaiseen tietotyypin käytttöpaikkaan, relaatiotietokantojen vastaava arvoalueiden (engl. *domain)* periaate on jäänyt vanhentuneeksi tutki-

muskohteeksi eikä ole saanut kovin merkittäviä käytännön toteutuksia.[14]

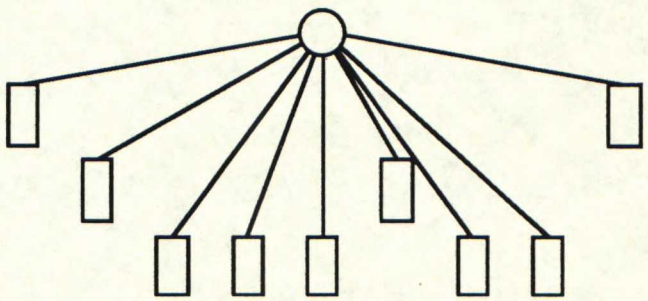
Useimmissa ohjelmointikielissä on siis käytettävissä tietuetyyppejä ja ohjelmointikielen kääntäjä pystyy avaamaan sisäkkäiset tietuetyypit ja tarkistamaan tehdyt kenttäviittaukset. Relaatiomallin mukaisissa käytännön tiedonhallintajärjestelmissä ei ole mitään vastaavaa, joten tyyppijärjestelmät täytyy sovittaa yhteen.

Yhteensovitus perustuu samalle periaatteelle, jolla ohjelmointikielen kääntäjä sovittaa sisäkkäin olevat tietueen kentät lineaariseen muistiin peräkkäin. Sisäkkäisyys on ainoastaan suunnitteluvaiheessa tarvittu ajattelun ja ylläpidettävyyden apukeino, joka voidaan toteutettaessa poistaa, "litistää" hierarkkinen rakenne vierekkäisistä kentistä koostuvaksi.



Kuva 3–7. Sisäkkäiset tietuetyypit.

Kuvassa 3–8 nähdään kuvan 3–7 sisäkkäiset tietuetyypit sisäkkäisyys poistettuna.

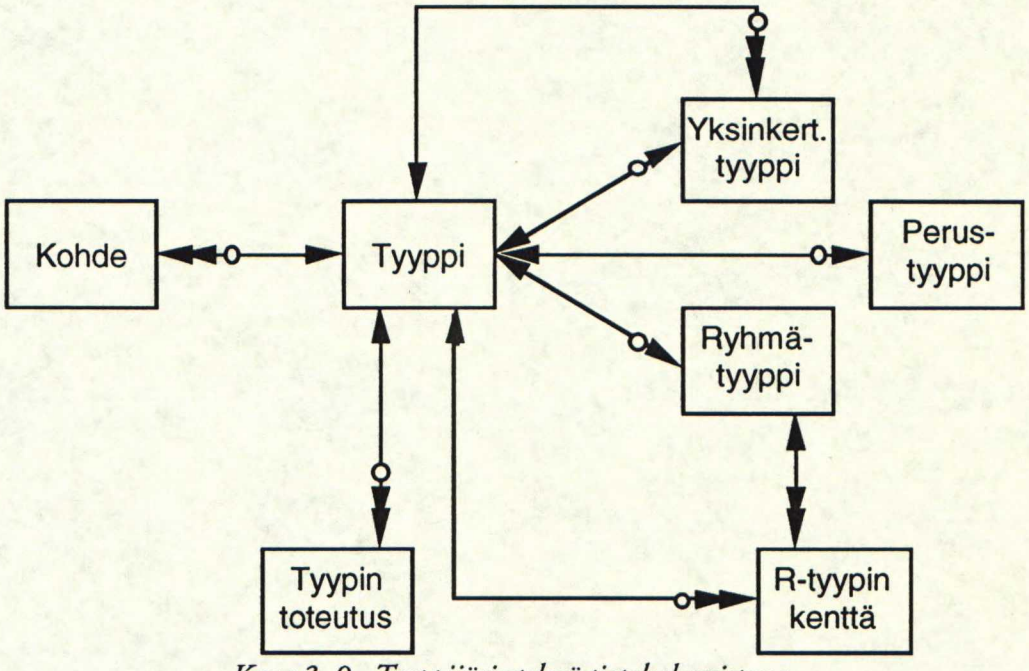


Kuva 3–8. Sisäkkäisyyden poisto.

Relaatiotiedonhallintajärjestelmien yhteydessä tällainen ajatus sopeutua koneellisesti johonkin ei-relationaaliseen, sisäkkäiseen tietokuvaukseen ei ole kovin yleinen. Melko lähelle samanlainen ajatus on lähteessä [Ceri 1991].

[14]Arvoalueita tutkittaessa ollaan oltu kiinnostuneempia monimutkaisten ja/tai käyttäjän itse määrittelemien tietotyyppien tuomisessa relaatioiden sarakkeiksi. kuin relaatiomallin vastaisten monikenttäisten tietuetyyppien mukaan ottamisesta.

Kuvassa 3–9 havainnollistetaan tietohakemistoon toteutettua tietotyyppijärjestelmää. Järjestelmä sisältää oleellisesti mahdollisuuden määritellä tietuetyyppejä sekä synonyymeja toisille tyypeille. Synonyymin annon yhteydessä tyypin toteutuskuvauksia voidaan muuttaa tarvituin osin.



Kuva 3–9. Tyyppijärjestelmä tietohakemistossa.

Tauluun "Perustyyppi" talletetaan niiden perustietotyyppien nimet, joita halutaan käyttää sovellusta tehtäessä ja joille ollaan valmiita tekemään sovitus kaikkiin käytettyihin sovellusvälineisiin. Tietohakemisto ei siis itsessään ota kantaa siihen, millaisia alkeistason tietotyyppejä on käytettävissä. Tavallisia ja useaan sovellusvälineeseen suoraviivaisesti toteutettavissa olevia perustietotyyppejä voisivat olla esimerkiksi "merkkijono, lukuarvo, raha, päivämäärä, kellonaika, prosentti" tms. Perustyyppi voi olla käytännössä monimutkainenkin ja toteutettu esimerkiksi olio-ohjelmointikielen luokkana. Perustietotyypin sovitusta relaatiotietokantaan on tosin vaikea tehdä, mikäli tyypin arvoa ei saa talletettua johonkin ko. relaatiotietokannan tarjoamista perustietotyypeistä yhdeksi sarakkeeksi.

Taulun "Yksinkertainen tyyppi" avulla saadaan rakennettua synonyymejä toisille tietohakemistoon jo määritellyille tyypeille. Yksinkertaiseen tyyppiin on varattu myös tilaa käytettävän (jo määritellyn) tyypin mahdollisesti tarvitsemille parametreille. Esimerkiksi voidaan luoda yksinkertainen tyyppi "postinumero", joka on perustyyppiä "merkkijono" parametreilla "k 5", joiden voidaan sopia tarkoittavan vaikkapa kiinteänmittaista viisimerkkistä merkkijonoa.

Taulupari "Ryhmätyyppi" ja "Ryhmätyypin kenttä" tallettavat rakenteiset eli tietuemuotoiset tyypit. Ryhmätyypin kentällä on ryhmätyypin sisällä yksikäsitteinen nimi ja myös järjestysnumero sekä viittaus tyyppiin, jota ko. kenttä on. Tavallisesti viittaus kohdistuu yksinkertaiseen tyyppiin tai toiseen ryhmätyyppiin. Mikäli perustyyppi ei tarvitse parametreja (esimerkiksi "päivämäärä" on tällainen), myös siihen voidaan viitata suoraan ryhmätyypin kentästä. Ryhmätyypin kenttään voisi ehkä lisätä tilaa tyypin parametreille.[15]

Tyyppijärjestelmän pääasiallinen käyttö on kuvata käsitemallin yksilötyyppien eli tietohakemiston kohteiden tietosisältö. Tavallisesti kohteen tietosisältöä kuvaava tyyppi on jokin ryhmätyyppi.

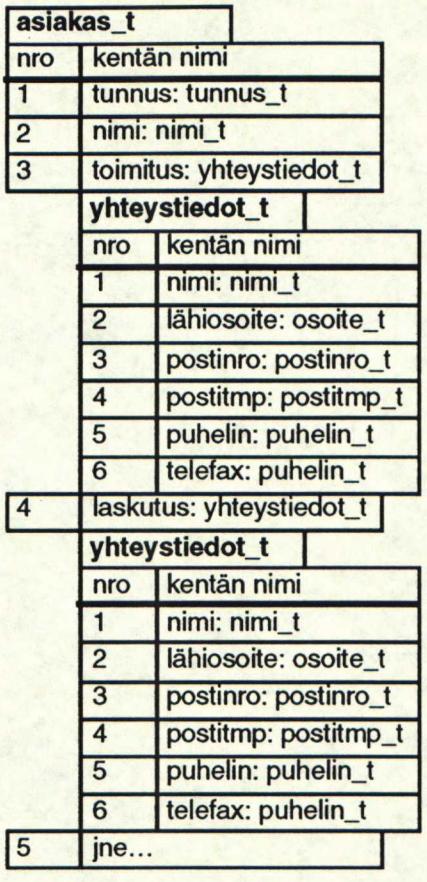
Sopivalla apuvälineellä voidaan käydä läpi jonkin ryhmätyypin alla olevat toiset tyypit (myös toiset ryhmätyypit) siten, että kohdatut perustyyppiset kentät kootaan yhteen luetteloon. Kohdetta kuvaavia (ryhmä)tyyppejä läpikäytäessä luonteva talletuspaikka tälle luettelolle on "Kohteen kentät"-taulu. Tässä yhteydessä samannimisille vierekkäisille kentille joudutaan keksimään sopivia alkuliitteitä osittain käsityönä.

Kuvassa 3–10 on esimerkki esiintulevista tilanteista purettaessa auki esimerkkikuvassa 2–6 esiintyneen "Asiakas"-taulun mahdollista ryhmätyyppinä kuvattua tietosisältöä. Huomataan, että lopputuloksena tulevassa pitkässä kenttäluettelossa on useita samannimisiä kenttiä. Ne joudutaan uudelleennimeämään ensimmäistä kertaa litistettäessä käsityönä, mutta annetut nimet voidaan taltioida kohteen kenttien tietoihin, jonne uudestaan litistettäessä vain päivitetään muuttuneet tiedot, tärkeimpinä perustyyppien parametrit.

Käyttöjä "litistetylle" kenttänäkemykselle on myös esimerkiksi käyttöliittymien puolella. Nykyiset käyttöliittymäohjelmistot osaavat toistaiseksi harvoin esittää rakenteisia ja sisäkkäisiä tietoja.

---

[15] Itse asiassa ensimmäisessä toteutetussa prototyypissä tyypin parametrit olivat aluksi myös ryhmätyypin kentällä. Ne poistettiin kuitenkin toisessa prototyypissä, sillä vaikutti siltä, että mikäli perustyyppi tarvitsee tietyt parametrit, useimmiten olisi hyvä määritellä uusi yksinkertainen tyyppi, jotta saataisiin talteen myöhempää ylläpitoa verten tieto, miksi juuri nuo parametrit tarvittiin sovelluksessa. Mikäli tyypin parametreille keksitään käyttöä muutenkin kuin perustyyppien yhteydessä, voidaan harkita parametrien palauttamista takaisin ryhmätyypin kenttään.



Kuva 3–10. Ryhmätyypin sisäkkäisyys.

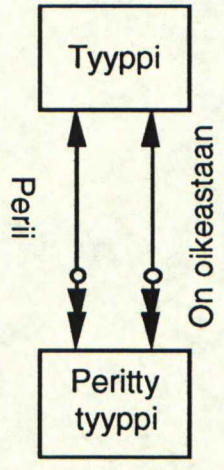
Sisäkkäisten määritysten avaaminen voidaan toteuttaa ilman erityistä ohjelmointia jaksossa 5.3, "Käsitemallin ja tietokannan muutokset" kuvatulla TIP-työkalulla käyttämällä sellaista pohjatiedostoa, joka noutaa sisäkkäiset tiedot TIP:n symbolitauluun ja vie litteän näkemyksen kentistä SQL:n avulla takaisin tietohakemistoon.

Esitetty tietotyyppijärjestelmä ei suoraan kykene tallettamaan vaihtuvasisältöisten tietojen kuvauksia (muunnelmatietueet). Lähteessä [Mujunen2 1991] olen esitellyt erilaisia vaihtoehtoja toteuttaa vaihtuvasisältöisiä tietueita relaatiojärjestelmillä. Esitetyistä vaihtoehdoista ne, joissa tukeudutaan eri muunnelmista luotaviin erillisiin tauluihin, ovat saatettavissa tälle tyyppijärjestelmälle sopivaan muotoon lisäämällä tauluja kuvaavia yksilötyyppejä jo käsitekaavioon ja samalla tietohakemiston kohteiksi.

Tietotyyppijärjestelmä ei suoraan tue myöskään tietotyyppien (lähinnä ryhmätyyppien) välistä perintää. Se voitaisiin kuitenkin lisätä jommallakummalla seuraavista kahdesta tavasta.

Mikäli tavoitteena on toteuttaa tietotyyppien välisestä perinnästä lähinnä instanssimuuttujien kopioituminen kantaisästä, voitaisiin ryhmätyypin kenttään kentän nimen sijaan merkitä, että kysymys on perivästä viittauksesta johonkin toiseen tyyppiin. Perittyä tyyppiä käsiteltäisiin siis kuten erikoisesti nimettyä ryhmätyypin kenttää, joka on ko. perittyä tyyppiä.

Toinen tapa kuvata tyyppien välinen perintä olisi antaa tyypin periä suoraan joukko kantatyyppejä. Tämä voitaisiin lisätä tyyppijärjestelmään kohteena "Perityt tyypit" seuraavasti (kuva 3–11):



Kuva 3–11. Perinnän lisääminen tyyppijärjestelmään.

Tällaista vapaata perintää on tietenkin hankalampaa käsitellä tietohakemistoa hyväksikäytettäessä, sillä se joudutaan ottamaan huomioon jokaisen tyyppiviittauksen yhteydessä.

Käyttämällä sellaista "Kohteen kentät"-taulua, jossa kohteen ja kentän tunnuksen ohella avaimeen on lisätty jokin versiotunnus saadaan talletuspaikka eri ajanhetkillä vallinneille tietokannan taulujen kenttäluetteloille. Tyyppijärjestelmään kohdistuneiden muutosten jälkeen voidaan tuottaa uusi versio kohteiden kenttäluetteloista, jolloin käytettävissä on sekä "uudet" että "vanhat" kohteen kentät. Uudet kentät kuvaavat toivotun kohdetilanteen ja vanhat kentät sen tilanteen, joka vallitsee toteutetussa tietokannassa juuri uusien kenttien luomisen jälkeen. Näiden kenttäluetteloiden vertailemiseen perustuvan ja muutokset toteuttavan SQL-lausejoukon automaattisempaa tuottamista käsitellään kohdassa 5.3, "Käsitemallin ja tietokannan muutokset".

Mikäli sopivia käyttötarkoituksia keksitään, versiotunnuksen voi tietenkin lisätä tarpeen mukaan muihinkin tietohakemiston tauluihin.

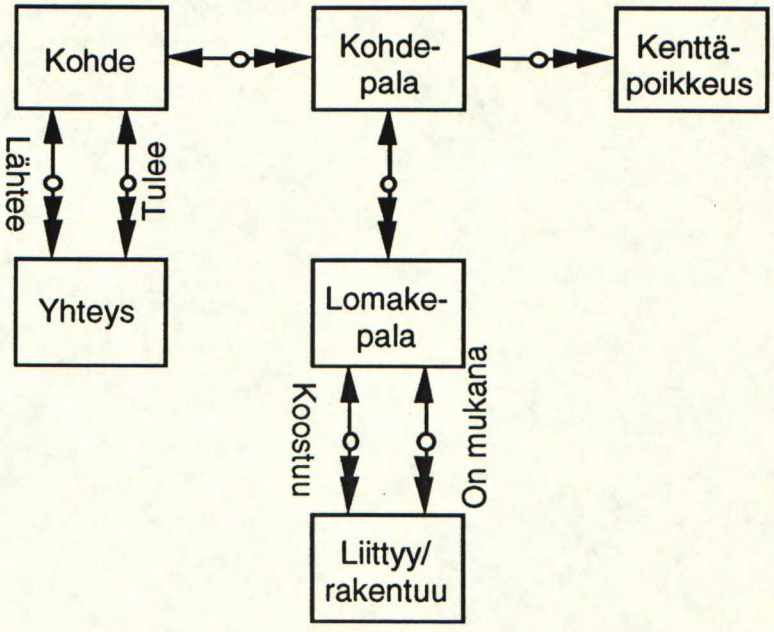
parametreissa) saatetaan peittää jokin tai täydentää jollekin perustyypille määriteltyä toteutusta. Tällainen toiminta täyttää kohdan 2.2, "Tietohakemisto", kuvassa 2–14 esitetyn tavoiteltavan kuvausten keskittämisperiaatteen. Samalla se demonstroi kuvausten keskittämisen monimutkaistavaa vaikutusta tietojen hyväksikäyttöön.

3.5.2 Tietohakemisto ja käyttöliittymä- ja tulostusohjelmistot

Kuvausten tiivistämiseksi ja kenttäviittausten lisäämiseksi sovelluksen tarvitsemat lomakkeet ja tulosteet pyritään kuvaamaan johtamalla niiden rakenne suoraan kohdassa 2.1.2, "Käsitemalli ja kaupallishallinnollisen sovelluksen rakenne" esitellyllä käsitemalliotteella. Otteesta muodostuva DAG kertoo suoraan, millä tavoin sisäkkäin käsitemallin kuvaamista tauluista haetut tiedot olisi sijoiteltava. Tietohakemistossa olevista "Yhteys"-taulun tiedoista saadaan vastaavasti tietää, esiintyykö alikohde pääkohteeseen nähden enintään kerran vai määräämättömän monta kertaa, ja tämän perusteella voidaan valita joko kiinteästi yhtä kohteen esiintymää tai vaihtoehtoisesti kooltaan vaihtelevaa esiintymäjoukkoa näyttämään/tulostamaan pystyvä esitystekniikka.

Mikäli käytetty käyttöliittymä- tai tulostusväline pystyy esittämään tiedon puuttumisen, myös yhteydeltä saatua vähimmäiskardinaliteettia "0" voidaan käyttää ilmoittamaan, että mikäli tieto puuttuu, puuttuminen halutaan näkyville.

Käsitemalliote-DAG:t voidaan tallettaa esimerkiksi seuraaviin tietohakemistotauluhin (kuva 3–13):



Kuva 3–13. Käsitemalliotteet tietohakemistossa.

Kukin sovelluksen kohde (eli tietokannan taulu) voi näyttäytyä lomakkeilla ja tulosteissa usealla eri tavalla, "palana kohteesta". Kohdepalaan kuuluvien kenttien luetteloa ei kuvata kiinteästi, vaan se muodostetaan kohteen kenttien ja palan yhteyteen talletettavien kenttäpoikkeusten avulla. Kohdepalaan nimittäin merkitään, tulevatko oletuksen mukaan kaikki kohteen kentät mukaan palaan vaiko vain erikseen mainitut. Taulua "Kenttäpoikkeus" käytetään luettelemaan ne kohteen kentät, jotka joko tulevat mukaan (mikäli palaan oletuksen mukaan ei oteta yhtään kenttää mukaan) tai jotka jätetään nimenomaan pois (mikäli palaan oletuksen mukaan otetaan kaikki kentät). Tällaisella menettelyllä voidaan kuvata mahdollisimman vähäisellä määrällä muutosaltista informaatiota molemmat tavanomaiset kohteiden esitystavat: kohteesta näytetään lähes kaikki kentät tai kohteesta näytetään vain tietyt kentät.

Käyttöliittymän kuvausosuudessa päädyttiin käyttämään tyyppijärjestelmän kuvausten sijaan kohteen litteää kenttäluetteloa. Pääsyynä tähän oli se, että nykyiset käyttöliittymävälineet toimivat harvoin rakenteisten tyyppien kanssa, vaan käyttävät tavallisesti yksittäisiä, alkeistasoisia kenttiä. Kohteen kuvaavien ryhmätyyppien mukaanotolla/poisjätöllä saatettaisiin saavuttaa mielenkiintoisen tiiviitä palojen kenttäkuvauksia, mutta tällöin myös käyttöliittymien puolella pitäisi tehdä oma ryhmätyyppien avaus.

Kohdepaloja käytetään materiaalina talletettaessa sovelluksen rakennetta eli lomakkeista ja tulosteista syntyviä käsitemalliote-DAG:ja. Lomake muodostetaan valitsemalla jonkin kohteen jokin pala lomakkeen pääkohteeksi ja liittämällä/ra-

kentamalla siihen toisia lomakepaloja. Näin talletetaan lomakkeen palojen sisäkkäisyys ja samalla lomakkeen käsittelemät kohteet (eli tietokantataulut).

Erityisen mielenkiintoista on nähdä, kuinka paljon käytännössä pystytään käyttämään hyväksi sellaisia lomakepaloja, jotka ovat mukana useissa toisissa lomakepaloissa. Periaatteessa tällä tavoin kuvaten voitaisiin sisällyttää vaikkapa kokonainen itsenäinen sovelluslomake osaksi toista lomaketta.

Nyt esitetty tietohakemistoratkaisu pystyy tallettamaan käyttöliittymien ja tulosteiden kuvaukset sellaiselle tasolle, jossa tiedetään kuhunkin kohdepalaan tulevat kentät sekä kohdepalojen väliset sisäkkäisyyssuhteet niiden näkyessä tietyissä lomakepaloissa. Yksittäisten kenttien sijoittelu tietyn kohdepalan sisällä sekä lomakepalojen sijoittelu vierekkäin sopiviin mittasuhteisiin ovat tehtäviä, jotka päätettiin jättää käyttöliittymävälineelle.

3.5.3 Tietohakemisto ja ohjelmointikielet

Koska tietohakemisto on määritelmänsä mukaan kuvauksia tiedon, ei toiminnan rakenteesta, on epätodennäköistä, että siihen olisi mielekästä säilöä varsinaista sovellusohjelmalogiikkaa tai fragmentteja tavanomaista 3GL-tyyppistä ohjelmakoodia. Sen sijaan tietohakemistoa voidaan käyttää ohjaamaan käytettävissä olevien lähdeohjelmapohjien valintaa ja niihin (tietohakemistoon kuvattujen kohteiden tietosisältöjen ohjaamina) tehtäviä muokkauksia.

Lähdeohjelmapohjien valintaa ohjaavat edellisessä kohdassa 3.5.2, "Tietohakemisto ja käyttöliittymä- ja tulostusohjelmistot", kuvatut tietohakemiston käsitemalliotteet. Otteen DAG-solmuja on toiminnallisesti kolmea päälajia: juurena oleva pääkohde, yksittäinen liittyvä tai rakentuva kohde ja joukko keskenään samanlaisia liittyviä tai rakentuvia kohteita. Kutakin päälajia varten voidaan laatia halutulla ohjelmointikielellä lajin kaipaamat toiminnot tietosisällöistä riippumatta hallitseva lähdeohjelmapohja käyttämällä hyväksi jaksossa 5 esiteltävää TIP-työkalua.

Lähdeohjelmapohjien käyttämistä sekä tietohakemiston yhteydessä että pelkästään TIP:n omin keinoin esitellään tarkemmin jaksossa 5, "Tietohakemiston hyväksikäyttö TIP-työkalun avulla".

Lähdeohjelmapohjien sisällä olevissa TIP-rakenteissa käytetään tietohakemistoa hyväksi mahdollisuuksien mukaan aina, kun on kysymys suoraan tai epäsuorasti tiedon rakenteeseen liittyvästä ohjelmakoodista. Tavallisia esimerkkejä tällaisesta ovat mm. tauluja ja lomakkeita vastaavien tietueiden kuvaukset, tietojen siirrot

em. tietueiden välillä ja tietojen muunnokset tiedon tyypin mukaan vaihtuvilla operaatioilla.

3.6 Yhteenveto laaditun tietohakemistoratkaisun rakenteesta

Liitteeseen 1 on koottu laaditun tietohakemistoratkaisun käsitemallin kaikki osat. Tästä käsitemallista on Tietosavo Oy:ssä prototyyppinä toteutettu Ingres-relaatiotietokantaan käsitemallin ja tyyppijärjestelmän osuudet. Koska käytettävissä ei ollut tekstimuotoisin kuvauksin toimivaa käyttöliittymäohjelmistoa, käsitemallin ote -osuutta ei voitu kokeilla. Tämän prototyypin ylläpito-ohjelmistosta on myös tehty Ingres:n 4GL-välinein kaksi prototyyppiä.

Prototyypistä saatujen käyttökokemusten perusteella tietohakemiston joustavuutta voisi pitää varsin hyvänä. Sen ja TIP-työkalun avulla on demonstroitu tietohakemiston käyttöä mm. Ingres:n SQL:n, C-kielen ja Ingres:n 4GL-kielen kanssa. Sen sijaan tietohakemiston ylläpito-ohjelmiston käytettävyyttä rajoitti varsinkin ensimmäisessä prototyypissä käytettyjen QBF- ja ABF-välineiden kankeus. Toinen prototyyppi osoittautui jo melko käyttökelpoiseksi, sillä sen toteuttamiseen käytettiin TIP-apuvälinettä Ingres:n 4GL:stä puuttuneiden aliohjelma- ja "include"-tyyppisten rakenteiden järjestämiseen ja näin pystyttiin järjestämään mm. joustavampia siirtymisiä eri ylläpitolomakkeiden välillä.

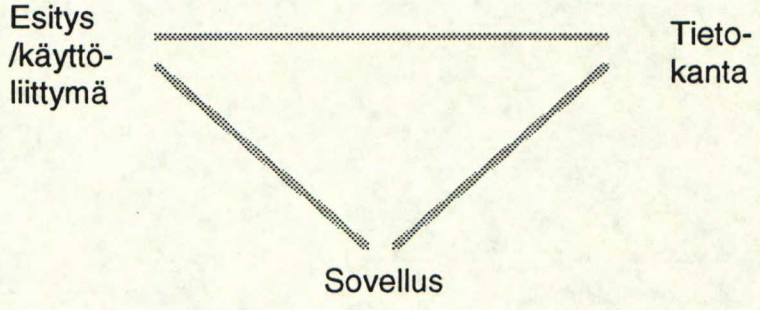
4 KAUPALLISHALLINNOLLISTEN SOVELLUSTEN TUOTTAMISEN JA YLLÄPIDON AUTOMATISOINTIKEINOJA

Johdannossa kaupallishallinnollinen sovellus määriteltiin laajaa tietojoukkoa vakioituneilla tavoilla käsitteleväksi järjestelmäksi. Tällaisella sovelluksella on tavallisesti laaja käyttäjäkunta, joka tarvitsee sovelluksen palveluja varsinaisen työnsä hoitamiseksi. Sovellusta käytetään eniten tapahtumankäsittelyyn (engl. *transaction processing),* jolla tarkoitetaan joukkoa yhtäaikaisia käyttäjiä tekemässä tosiajassa määrämuotoisia tietokantaoperaatioita.

Kaupallishallinnolliset sovellukset ovat tietopainotteisia: keskeisiä käyttökohteita ovat tietojen arkistointi ja etsintä — sitävastoin laskentaa, päättelyä tms. varsinaista "käsittelyä" niissä on tiedon siirtelyn määrään nähden melko vähän.

Sovellus rakentuu suurelle joukolle keskenään samankaltaisia ja tasa-arvoisia ohjelmia, lomakkeita ja tulosteita. Lomakkeet tarjoavat arkistotoimintojen ja paperilomakkeiden korvaamisen ohella myös tietokannassa navigointimahdollisuuksia. Usein tarjolla oleva käyttöliittymävälineistö ohjaa liittämään navigointiin helpon siirtymisen lomakkeelta toiselle. Tulosteisiin halutaan vapaasti valittavia hakuehtoja, lajittelujärjestyksiä ja mielellään myös ulkoasuja.

Seuraava kuva 4–1 esittää kaupallishallinnollisen sovelluksen toteutusrakennetta käytettäessä tavanomaisia sovelluskehitysvälineitä erityisesti etsittäessä mahdollisuuksia hajottaa sovelluksen suoritus useampaan tietokoneeseen. Kolmiosaisesta sovelluksen rakenteesta seuraa myös sovellukseen tehtävien muokkausten hajaantuminen kolmelle taholle siten, että kaikilla tahoilla on toistensa kanssa osittain yhteisiä osia.



Kuva 4–1. "Kaupallishallinnollinen kolmiyhteys".

4.1 Systeemityömalleista

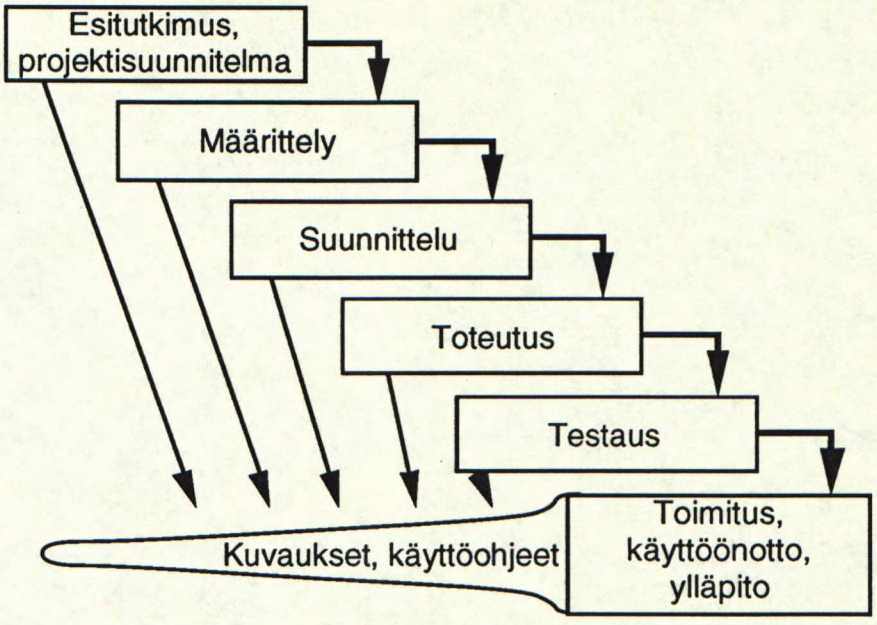
Sovellusten tuottamiseen tähtäävää systeemityötä on kuluneiden vuosikymmenten aikana koetettu jäsentää monilla eri tavoilla. Tavoista suosituimmiksi ovat yltäneet erilaisiin vaihejakoihin perustuvat systeemityömallit. Myös prototyyppien asteittaiseen kehittämiseen perustuvaa lähestymistapaa käytetään laajalti.

Sovellustyön jäsentäminen on välttämätöntä tuotantotehtävän saattamiseksi hallittaviin osiin. Ilman järkevää ositusta sovelluksen parissa työskentelevälle ihmisjoukolle on vaikeaa jakaa tehtäviä siten, että niiden eteneminen pysyisi hallinnassa ja olisi koordinoitua. Onnistuneen osituksen hallitulle toteuttamiselle välttämätön edellytys on sujuva työryhmän sisäinen kommunikaatio. Sen toimimiselle perustuu sekä työtehtävien joustava siirtyminen ryhmän jäseneltä toiselle että tietojen ja kokemuksen leviäminen organisaatiossa. Kommunikointitaitojen ja -menettelytapojen kehittämiseen systeemityömallit ottavat yllättävän vähän kantaa.

4.1.1 Vaihemalli

Vaihemallin käyttö systeemityössä tarkoittaa työn jakamista toinen toisiaan seuraaviin vaiheisiin. Muita nimityksiä mallille ovat vesiputous- ja elinkaarimalli.

Kuva 4–2 esittää systeemityön etenemistä vaihemallissa. Pohjana kuvassa on käytetty mm. lähteiden [DeMarco 1979] ja [IBM2 1991] vaihejakoja, mutta vaiheiden nimiä ja merkityksiä on muokattu hieman yhdenmukaisemmiksi Tietosavo Oy:ssä käytetyn vaihejaon kanssa.



Kuva 4–2. Systeemityön vaihemalli.

"Vesiputous"-nimityksen malli on saanut kuvassa 4–2 selvästi näkyvästä työn tulosten vesiputousmaisesta virtaamisesta vaiheesta toiseen. Mallissa odotetaan tiukasti, että seuraava vaihe voi alkaa vasta, kun edellisen vaiheen työn tulokset ovat kaikki valmiina ja näin materiaalina seuraavan vaiheen käytettävissä. Tämä on mallin muunnelmissa yhdenmukaisimpana säilyvä yhteinen piirre. Vaiheiden lukumäärä, järjestys, nimet ja asiasisältö sen sijaan vaihtelevat jonkin verran. Seuraavassa kuvailen kuvassa esiintyvien vaiheiden sisältöä.

Esitutkimusvaiheessa varmistetaan tavoitteet, joihin uudella järjestelmällä pyritään, kirjataan reunaehdoiksi ne vaatimukset, jotka järjestelmän on kyettävä täyttämään sekä kerätään esiintulevia tarpeita lähtökohdaksi jatkotyölle. Tärkeänä osana esitutkimusta kartoitetaan, onko ylipäänsä mielekästä ja kannattavaa lähteä suunnittelemaan ja toteuttamaan aiottua sovellusta. Mikäli vastaus vaikuttaa myönteiseltä, kirjataan päätökseen vaikuttaneet perustelut projektisuunnitelmaksi, jossa otetaan kantaa järjestelmän toteuttamiseen liittyviin resurssikysymyksiin, joita ovat taloudelliset, aika-, työntekijä-, laite- ja muut resurssit.

Määrittelyvaiheessa järjestetään kerätyt tarpeet tärkeysjärjestykseen ja niitä sekä tavoitteita ja vaatimuksia materiaalina käyttäen laaditaan järjestelmälle karkea hahmo. Apuna tässä voidaan käyttää esimerkiksi SA-menetelmää: menetelmässä alkuvaiheessa selvitetään se ympäristö, jossa järjestelmä tulee toimimaan (eri osapuolet, joiden kanssa järjestelmä on vuorovaikutuksessa) sekä järjestelmän toiminnan karkea ositus. Myös järjestelmän pysyvistä tiedoista kannattaa tehdä karkean tarkkuuden käsitemalli, jota myöhemmissä vaiheissa tarkennetaan.

Suunnitteluvaiheessa käsitemalli tarkennetaan kattamaan mieluiten kaikki tarvittavat tietokantaan talletettavat kohteet. Myös käsitemallin yksilötyyppien tietosisältö pyritään kuvaamaan. Järjestelmältä vaadittavat toiminnot tarkennetaan kukin määrittelyvaiheessa löydetty osa-alue kerrallaan vähintään sellaiselle tarkkuustasolle, että kuvausten perusteella toteutusvaiheessa toteuttajat pystyvät laatimaan tarvitut sovellusohjelmat. Perinteisesti tämä tarkoittaa mm. näyttö- ja tulostemallien laatimista, ohjelmien moduulirakenteen suunnittelua sekä SD-kaavioiden viemistä niin pitkälle, että niissä näkyy jo yksittäisten ohjelmien sisäistä toiminnallista rakennetta.

Toteutusvaiheessa tehdyt suunnitelmat muunnetaan käytettyjen sovellusvälineiden (ohjelmointikielten, tiedonhallintajärjestelmien jne.) vaatimiksi kuvauksiksi. Periaatteessa suunnitelmien pitää olla tällöin niin kattavia, että muuntamisessa on kysymys lähes mekaanisesta työstä. Toteutettu sovellus testataan vaatimuksia, tavoitteita, tarpeita sekä suunnitelmadokumentteja vastaan vertaamalla, toteuttaako sovellus halutut ja suunnitellut tehtävät. Mikäli sovellus vastaa suunnitelmaa, se toimitetaan sovelluksen käyttäjäasiakkaille ja otetaan käyttöön.

Käyttöönoton jälkeen sovelluksen elinkaaressa seuraa ylläpitovaihe, jota kestää koko sovelluksen käyttöiän ajan. Siinä sovelluksen käytöstä syntyneitä kehitystarpeita työstetään suunnitelmien läpi toteutuksen korjaukseen, ja lopputuloksena saadaan tarkemmin tarpeita vastaava sovellus ja (toivottavasti myös) suunnitteludokumentit. Juuri ylläpidon sisältävistä vaihemalleista puhutaan joskus elinkaarimalleina. Toisissa vaihemallin muunnelmissa sovellus nähdään "valmiina" silloin, kun se vastaa suunnitteluvaiheen vaatimuksia ja ylläpitoa ei tunnusteta olevan olemassa. Ylläpidon sijaan ajatellaan, että koko sovellus laaditaan uudelleen vaihemallia hyväksikäyttäen sellaisissa tilanteissa, joissa sovelluksen laadinnan perusteena olleet tavoitteet, tarpeet ja vaatimukset ovat muuttuneet niin paljon, että syntynyt lopputulos ei enää vastaa tosiasiallisesti haluttua.

Vaihemallin keskeisiin periaatteellisiin etuihin kuuluu järjestelmällisyys: syntyneistä työn tuloksista voidaan suoraan nähdä, mihin vaiheeseen sovelluksen tuotanto ja valmiusaste ovat edenneet. Tehtävällä sovellustyöllä on tukipisteenään edellisen vaiheen tulokset, joihin nojautumalla kaikelle tehtävälle työlle on esittää perustelu, miksi sitä ollaan tekemässä.

Se, että myöhemmissä vaiheissa tarkennetaan edellisten vaiheiden tuloksia, aiheuttaa helposti saman työn tekemistä toistuvasti lähes uudelleen. Tämän välttämiseksi on oleellista, että suunnitelmien taltiointiin käytetyt välineet tukevat suunnitelmien kehittämistä edelleen mieluiten siten, että aiempien vaiheiden tuloksista voidaan lainata osia työn alla olevan vaiheen materiaaleihin niin, että aiempaa kuvausta muutettaessa muutos päivittyy automaattisesti myös uudempaan kuvaukseen.

Jos vaihemallin varhaisvaiheessa tehdään virhe eikä sitä huomata välittömästi, virheen vaikutus kertautuu nopeasti vaiheiden edetessä. Tämä johtuu siitä, että seuraavat vaiheet perustuvat aina edellisten vaiheiden työlle. Virheet vaihemallissa korjataan palaamalla vaiheissa taaksepäin tekemään uudestaan se kohta, jossa alkuperäinen virhe syntyi; ellei näin tehdä, suunnitelmadokumentit vanhenevat. Tämä paluupakko tekee vaihemallista joskus raskaan ylläpidettävän, koska muutostarpeet joudutaan periaatteessa viemään läpi koko vesiputouksen.

4.1.2 Prototyyppimalli

Prototyyppimallissa lyhennetään sovelluksen matkaa määrittelystä toteutukseen. Sovelluksesta pyritään tekemään mahdollisimman pian alustavien määritysten jälkeen prototyyppi, joka havainnollistaa järjestelmän tulevaa toimintaa. "Todelliselta" vaikuttavasta järjestelmästä näytetään pystyvän löytämään korjattavaa ja parannettavaa huomattavasti helpommin kuin pakostakin abstrakteiksi jäävistä suunnitteludokumenteista. Muutostarpeiden perusteella prototyyppiä korjataan ja täydennetään tarkemmaksi malliksi halutusta sovelluksesta. Vaarana tällaisessa "suoraohjauksessa" on se, että muutostarpeita ei osata asettaa toteutuksessa tärkeysjärjestykseen, vaan alkuvaiheessa sorrutaan käsittelemään toissijaisia muutosehdotuksia.

Prototyyppimalli voidaan nähdä myös vaihemallissa nähtävillä olevan korjauskäytännön "virallistamisena". Sovelluksessa todetut puutteet vain eivät johda määrittely-suunnittelu -prosessin käynnistymiseen, vaan puute koetetaan suunnitella ja toteuttaa lähes suoraan prototyyppiin. Merkittäväksi pulmakysymykseksi prototyyppimallin soveltamisessa muodostuu tällöin käytettävien sovellusvälineiden valinta. Mikäli väline sopii hyvin ensimmäisten prototyyppien nopeaan laadintaan, on todennäköistä, että näitä prototyyppejä ei pystytä muokkaamaan niin paljon kuin olisi tarpeen prototyypin saattamiseksi lopullisen sovelluksen tasoiseen kuntoon. Mikäli taas välineellä voidaan laatia tuotantokelpoinen sovellus, on hyvin todennäköistä, että ensimmäisten prototyyppien tuottamiseen joudutaan käyttämään kohtuuttoman paljon aikaa ja työtä ennen kuin päästään arvioi-

maan tarpeeksi todelliselta vaikuttavaa prototyyppisovellusta.

Toteutustyön edistyessä sovelluksen "dokumentteina" kerääntyy mahdollisesti muistiinpanoja asioista, jotka pitäisi muuttaa kullakin hetkellä kokeiltavana olevassa prototyypin versiossa. Varsinaista sovelluksen rakennetta selvittävää suunnitelmadokumentaatiota ei muodostu.

Prototyyppilähestymistapaa käytettäessä saatetaan törmätä myös tehdystä muutoksesta johtuvien seurannaismuutosten määrän räjähdysmäiseen kasvuun. Mikäli näin käy, joudutaan tasapainoilemaan sen suhteen, kumpi tulee kalliimmaksi, hylätä jo kerran tehty työ tekemällä osa prototyypistä kokonaan uudelleen vai yrittää korjata vaikeasti muutettava prototyyppi toimivaksi.

Sekä vaihemallilla että prototyyppilähestymistavalla on omat etunsa. Tuntuisikin luontevalta koettaa yhdistää mallien parhaat puolet. Tällaista yritetään seuraavassa kohdassa.

4.2 Muunneltu elinkaarimalli

Tutkitaan hieman tarkemmin vaihemallin kulkua häiritseviä tekijöitä. Tarkoituksena on löytää vaikeimmin poistettavat häiriötekijät, jotta vaihemallia voitaisiin kehittää sietämään niitä.

Kokemusteni mukaan tavallisimmat syyt siihen, että vaihemallia käytettäessä toteutus- ja testausvaiheista joudutaan palaamaan taaksepäin määrittelyyn ja suunnitteluun eli käyttämään prototyyppimallia, ovat seuraavat:

1) Merkittävä puute tai käsittelemättä jäänyt kohta määrityksissä ja/tai suunnitelmissa.

2) Pienehkö puute tai avoimeksi jätetty kohta suunnitelmissa, jota ammattitaidoton tai sovellusaluetta tuntematon toteuttaja ei ole kyennyt itsenäisesti tarkentamaan.

3) Suoranainen virhe määrityksissä ja/tai suunnitelmissa.

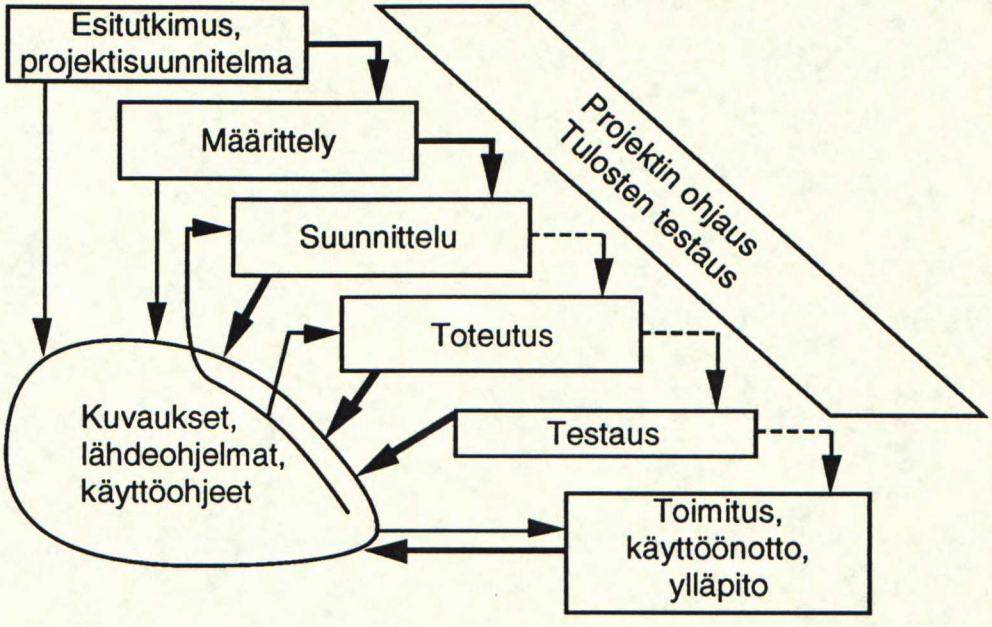
Perimmältään kaikissa näissä virhesyissä on taustalla kommunikointiongelma: kommunikaatiokatkos sovellusasiakkaan, suunnittelijoiden ja/tai toteuttajien kesken. Suunnittelija ei ymmärrä sovelluksen tarvitsijan kuvailemaa tarvetta ja tilannetta, suunnittelijat eivät pysty omaksumaan ja välittämään toisilleen eteenpäin sovelluksesta muodostettuja käsityksiä eikä sovelluksen toteuttaja löydä

suunnitteludokumenteista (tai edes suoraan suunnittelijoilta) toteutukseen tarvitsemiaan tietoja.

Ensimmäisen lajin merkittävät poisjäännit aiheutuvat tavallisesti ihmisen kyvyttömyydestä hahmottaa nopeasti laajalle yksityiskohtajoukolle sellaista rakennetta, jonka avulla kaikki yksityiskohdat muistettaisiin käydä läpi. Toisen lajin suunnitelman vajavaisuus johtuu usein toteuttajan taitojen ja tietojen yliarvionnista, mutta usein myös siitä valtavasta määrästä tietoja ja yksityiskohtia, jotka pitäisi ottaa yhtäaikaisesti huomioon toteutussuunnitelmaa tehtäessä. Tällaisessa suoritustilanteessa ihmisellä on taipumus ja käytännön järjen sanelema pakko jättää "eniten itsestäänselviltä vaikuttavia" asioita pois. Kolmannen lajin varsinaiset virheet johtuvat kommunikointikatkosten ohella mitä moninaisimmista syistä: henkilökohtaisista periaateristiriidoista, riittämättömistä tai virheellisistä pohjatiedoista, välinpitämättömyydestä, väärinkäsityksistä, pienimmän välittömän työmäärän etsimisestä jne.

Kuten nämä virhemekanismien hahmotukset sekä vaihemallista saadut käytännön kokemukset osoittavat, ei ole realistista odottaa, ettei tiukasti vaiheittain edenneen sovellustyön lopputuloksessa olisi koskaan korjattavaa. Korjaukset kohdistuvat niin sovelluksen käsittelemien tietojen rakenteeseen kuin itse käsittelyihinkin. Pieniä tietojen muutoksia tarvitaan varsin usein, mutta silti niiden vaikutukset ulottuvat tavallisesti laajalle ympäri toteutusta. Suuret, periaatteelliset toimintojen muutokset vaikuttavat myös moniin ohjelmiin, mutta kuitenkin kaikkiin samantyyppisiin ohjelmiin samalla tavalla. Pienet, mutta usein esiintyvät toimintojen muutokset ovat onneksi hyvin lokalisoituneita: on kysymys juuri tietyn, ainutkertaisesti esiintyvän toiminnan muuttamisesta.

Näistä muutosten luonteenpiirteistä lähtien koetetaan muodostaa systeemityömallia, joka sallii prototyyppimäisen lähestymistavan myös pienten tietomuutosten ja suurten toimintaperiaatteiden muutosten suhteen. Luonnos tällaisesta on kuvassa 4–3 esitetty muunneltu elinkaarimalli.



Kuva 4–3. Muunneltu elinkaarimalli.

Mallin esitutkimus- ja määrittelyvaiheet pyritään toteuttamaan alkuperäisen vaihemallin tapaan kokonaisina. Näissä vaiheittainen toteutus onkin myöhempiä vaiheita paremmin mahdollista, sillä käsiteltävä tietomäärä on täällä selvästi suunnittelua ja toteutusta suppeampi. Vaihemallin yhteydessä käytetyt SA/SD-menetelmät vaikuttavat myös aivan sopivilta ylätason hierarkkisen jaon löytämiseen.

Luonnollisesti myös suunnittelussa, toteuttamisessa ja testauksessa toimitaan jossain määrin vaiheittain. Kuitenkin lienee selvää, että esimerkiksi koko suunnitelman valmistumista ei jäädä odottamaan ennen ensimmäisten sovellusosien toteuttamisen aloittamista. Suunnittelusta "irronneita" pieniä osa-alueita (esimerkiksi muutaman käsitemalliotteen ryhmiä) siirretään toteutukseen sitä mukaa kun niitä saadaan valmiiksi. Toteutuksessa pyritään käyttämään 2–3 henkilön pienryhmiä kunkin osa-alueen parissa. Pienellä ryhmällä varmistetaan, että ryhmän sisäinen kommunikointi voisi toimia tehokkaasti. Ryhmää käytetään mieluummin kuin yksittäistä toteuttajaa siksi, että ryhmä ei lukkiudu ongelmatilanteeseen pohtimaan sitä pitkiksi ajoiksi, vaan pyrkii kollektiivisesti etsimään ongelmaan ratkaisun. Sopivassa ilmapiirissä ryhmässä syntyy myös lievää kilpailua siitä, kuka selviää toteutustehtävistä sujuvimmin ja nopeimmin.

Tietohakemiston avulla pyritään saamaan prototyyppimäinen työskentely kehitystyön alla olevan ohjelmiston parissa laajenemaan pelkän lähdekoodin muokkailusta kokonaisten kuvausten muokkailuun. Kun lähdeohjelmat, tietokannan luonti- ja käsittelylauseet, käyttöliittymäkuvaukset ja dokumentaatio (tärkeim-

pänä käyttöohjeet) on sidottu tietoriippuvaisilta osiltaan yhteiskäyttöiseen tietohakemistoon, niitä ei ole mielekästä muuttaa muuten kuin tietohakemiston avulla, jolloin varmistetaan muutosten leviäminen kaikkiin toteutusosiin. Periaatteellisten toimintomuutosten koneellistettuun levittämiseen pureudutaan eristämällä periaatteellisesti samanlaiset ohjelmaosat lukumääräisesti mahdollisimman harvoihin lähdeohjelmapohjiin TIP-tyyppisen työkalun avulla.

Kaikella tällä toiminnalla toivotaan päästävän siihen, että protyyppilähestymistavan käyttö tulisi "halvemmaksi", kun muutospäätöksestä ei seuraa välittömästi suurta määrää käsityötä. Prototyyppimallin haittapuolena ollutta suunnitteludokumenttien muodostumisen puutetta koetetaan kompensoida sillä, että toteutusvaiheessa pyritään käsittelemään mahdollisimman "suunnittelusuuntautuneita" kuvauksia. Voidaan puhua aktiivisista suunnitelmista, joiden tahdissa varsinainen toteutus elää enemmän tai vähemmän automaattisesti. Tällaisia aktiivisia suunnitelmanosia ovat esimerkiksi kohdassa 3.5.2 esitellyt tietohakemistoon talletetut käsitemalliotteet.

Tietyn suunnitelmanosan "aktivoitumisen" tärkein edellytys on se, että ko. osan muokkaaminen aiheuttaa konkreettisia hyötyjä toteutuksen puolella. Esimerkiksi käsitekaavion piirtämisen yhteydessä tämä tarkoittaa sitä, että piirtämällä uusi yksilötyyppi kaavioon pitäisi saada aikaan vähintään yksilötyyppiä vastaavan kohteen perustuminen tietohakemistoon. Mikäli on hiemankaan helpompaa ja nopeampaa perustaa yksilötyyppi suoraan vain tietohakemistoon, sovelluksen tekijät myös tekevät näin.[16]

4.3 Keskeisiä automatisointikohteita elinkaarimallissa

Seuraavissa kohdissa tuon esiin tärkeimpiä muunnellussa elinkaarimallissa automatisoitavissa olevia muutostilanteita. Automatisoinnin "tärkeys" on arvioitu periaatteessa muutostarpeen toistumistiheyden ja muutoksen aiheuttaman käsityömäärän tulona.

4.3.1 Tietosisällöistä riippuvat automatisoinnit

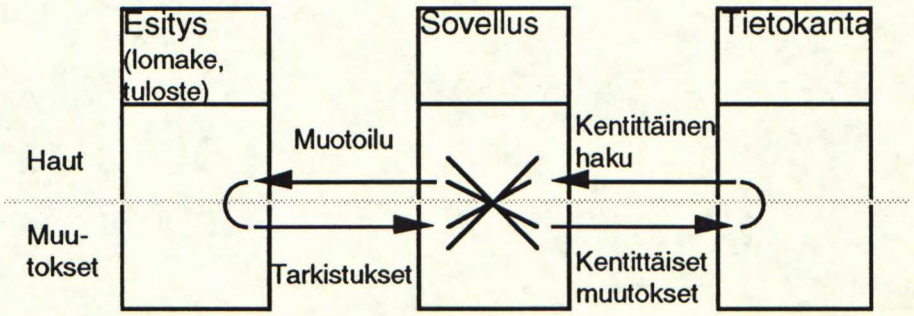
Talletettavien tietojen muuttuminen on tavallisin muutostilanne kaupallishallinnollisissa sovelluksissa. Yleisin muutos on jonkin yksilötyypin tietosisällön muuttuminen. Toiseksi yleisin on yksilötyyppien muuttuminen (uusien yksilötyyppien lisäys, vanhojen poisto). Tietojen rakenteen muuttaminen säteilee eniten

[16]Varmaan myös vastakkaista lähestymistapaa voisi käyttää: kohteen lisääminen tietohakemistoon aiheuttaa yksilötyypin ilmestymisen käsitekaavioon. Oleellisinta on, että tiettyä muutosta ei tarvitse tehdä käsin kuin yhteen paikkaan.

läpi käytettyjen sovellusvälineiden: sovellusohjelmat, käyttöliittymät, tulosteet, tietokannat, dokumentit ja käyttöohjeet muuttuvat sekä tietokantojen vanhoille tietosisällöille tarvitaan konversiot. Tästä syystä tietojen muuttamisen automatisointi on tärkein automatisointikohde.

Jotta tietojen muutokset saataisiin levitettyä automaattisesti laajalle, täytyy tuottamistekniikoita soveltaa läpi mahdollisimman monen sovellusvälineen.

Kuvasta 4–4 nähdään, kuinka tavanomaisen lomakemuotoisen sovellusohjelman keskeinen toiminnallisuus on yksittäisten tietokenttien tasolla tapahtuvaa uudelleenjärjestelyä, muotoilua ja suodatusta.



Kuva 4–4. Kentittäinen uudelleenjärjestely.

SQL sanelee tietojen siirtelyn tapahtuvan kentittäin lähellä relaatiotietokantaa. Lisäksi SQL:n rakenteet vaativat paljon kenttäluetteloiden toistoa: jopa lisäys- ja muutoslauseiden tapauksessa samassa lauseessa tarvitaan käsiteltyjen sarakkeiden luettelo kahteen kertaan. Nämä osat voidaan tuottaa kohteen kenttien ohjaamana.

Muotoilu- ja tarkistuspuolella käsiteltävien kenttien perustietotyypit joudutaan ottamaan huomioon ja tuottamaan niiden mukaan vaihtuvaa sovelluskoodia elleivät sekä sovellusohjelmointikieli että käyttöliittymäväline tue polymorfisia ja/tai ylimääriteltyjä (engl. *overloading)* muotoilu- ja tarkastusrutiineita. Lomakkeen kenttien luettelot saadaan johtamalla ne lomakkeella esiintyvien kohteiden kentistä ja kohdepalojen kenttäpoikkeuksista kohdassa 3.5.2, "Tietohakemisto ja käyttöliittymä- ja tulostusohjelmistot" esitetyllä tavalla

Suoraan käsitemallista kohdassa 3.5.2, "Tietohakemisto ja käyttöliittymä- ja tulostusohjelmistot" kuvatulla tavalla (ote käsitemallista) johdetut toiminnot ovat sovellusosiltaan melko suoraviivaisia (uudelleen-)tuotettavia tietokannan rakenteen muuttuessa. Jos käytettävä käyttöliittymäväline pystyy mitoitamaan kentät kuvaruudulle jonkin verran omatoimisesti, myös käyttöliittymäkuvaukset saadaan rakennettua uudestaan ilman käsityötä, ellei kenttien lukumäärä muutu

oleellisesti.

4.3.2 Vakiotoimintojen automaattinen järjestäminen

Vaikka itse vakiotoiminnot eivät riipukaan suoraan tietohakemiston sisällöstä, on niiden koneellistettu tuottaminen edellytys, jotta edellisessä kohdassa 4.3.2 esitetty tietomuutoksiin sopeutuminen voisi onnistua. Toteuttamalla automaattinen tietomuutoksiin sopeutuminen saavutetaan samalla kuitenkin se, että vakiotavalla toimiva osa sovellusohjelmasta on kuvattu keskitetysti yhdessä paikassa, ko. lajisen vakiotoiminnon ohjelmapohjassa. Nyt jos vakiotoimintoihin halutaan muutoksia, muutos leviää automaattisesti kaikkiin samankaltaisiin ohjelmiin.

Kriittiseksi tekijäksi toimintojen monistamisen onnistumisessa nousee se, että käsin tehdyt ei-vakiot toiminnot saadaan rajoituksetta lomitettua vakiotoimintojen joukkoon. Tähän päästään vain suunnittelemalla ohjelmapohjiin riittävästi paikkoja, jonne käsin tehtyä koodia voi tuottovälineen avulla lomittaa. Sellaisissa harvinaisissa tapauksissa, joissa vakiotoiminnot soveltuvat huonosti, voidaan tietojen rakenteen muutoksilta suojautua käyttämällä myös "ainutkertaisissa" sovellusosissa tiedon kuvausten hakuja tietohakemistosta. Tällaiset lähdeohjelmat eivät tietenkään pysty mukautumaan edes sellaisiin vakiotoimintojen muutoksiin, joiden asiallisesti ottaen pitäisi vaikuttaa myös ko. ohjelman rakenteeseen.

4.3.3 Lomakkeiden välisten siirtymisten automaattinen järjestäminen

Ensimmäisenä edellytyksenä automaattisten lomakkeiden välisten siirtymisten järjestämiselle on, että kaikkialla, missä on esillä tietokannasta haettuja tietoja, pitäisi olla selvillä kaikki näiden tietojen sisään upotetut vierasavaimet. Vierasavaimista pitäisi myös pystyä päättelemään ne kohteet, joihin ko. vierasavaimilla olisi "pääsy". Jaksossa 3 esitellyssä tietohakemistototeutuksessa ei ole varauduttu kuvaamaan vierasavaimia (eikä pääavaimia) joustavammin kuin upottamalla ne kohteen toteutuskuvaukseen.

Vierasavaimia vastaavien kohteiden avulla voidaan etsiä tietohakemistosta kaikki ne lomakkeet, joissa nuo kohteet ovat pääkohteiden roolissa. Tämä etsintä on todella hankalaa tehdä käsityönä, sillä uutta lomaketta lisättäessä ei mitenkään pysty kuvittelemaan, mistä kaikkialta siihen voitaisiin tulla. Jotta uuden lomakkeen lisääminen ei aiheuttaisi muutoksia vanhoissa lomakkeissa, lienee parasta hakea kohdetta vastaavat lomakkeet sovelluksen ajonaikaisesti. Tätä varten ei välttämättä tarvitse järjestää koko tietohakemistoa ajonaikaisesti saataville, vaan

riittää, että sovelluksen tuottamisen yhteydessä tuotetaan "Kohteen lomakkeet"-tyyppinen taulu siirtymistietoja varten.

Yhteyksien järjestämisellä järjestetään sovelluksen loppukäyttäjälle yhtenäinen "tie eteenpäin": samalla menettelyllä hän tietää saavansa näkyville kaikki vähänkään järkevät jatkomahdollisuudet. Sovelluksen tekijän kannalta vältetään saman käsittelytaidon monistaminen useaan saman sovelluksen lomakkeeseen. Automaattisesti tuotettavien vakiotoimintojen osalta tämä ei ehkä olisi kovin tärkeää, mutta käsin tehtyjen, sovelluslogiikkaa sisältävien osien monistamista pitää välttää.[17]

Runsaasti tietoa sisältävästä lomakkeesta päästään automaattisin siirtymisin varsin moneen kohteeseen, koska siinä on esillä useita vierasavaimia Lisäksi monista kohteista on olemassa helposti kymmeniä lomakkeita, jossa ko. kohde on pääkohteena. Suurten määrien takia voidaan joutua järjestämään lomakeluettelolle jokin oivaltava suosituimmusjärjestykseen asettelu tai dynaamisesti laadittu esitystapa käyttöliittymän puolella. Eri lomakkeiden suosituimmuutta voitaisiin tilastoida ja käyttää tilastoja lomakkeiden karsinnan perusteena. Dynaamisesti tarkentuvat valikot pystyvät pakkaamaan pieneen ruututilaan lukuisia vaihtoehtoja. Tällaisilla valikoilla periaatteena on tarjota perinteinen hierarkkinen valikkorakenne tasot vierekkäin sijoiteltuina. Lomakkeiden määrää supistaa myös tehokkaasti, mikäli käyttäjä osoittaa lomakkeelta sitä vierasavainta, johon liittyviä lomakkeita hän haluaisi nähdä..

4.3.4 Dokumenttien täydentäminen

Dokumentaation koneellistetulla tietohakemiston tiedoin täydentämisellä ei saavuteta yhtä suurta työnsäästön hyötysuhdetta kuin edellä kuvatuissa tapauksissa. Taululuetteloiden, kenttäluetteloiden, lomakeluetteloiden ja vastaavien haku dokumenttien ja käyttöohjeiden joukkoon ei varsinaisesti aktivoi niitä sellaiseksi osaksi sovelluskuvausta, jonka muutos vaikuttaisi suoraan toteutukseen. Sen sijaan se ehkä madaltaa dokumentaation tekokynnystä, koska tekijän ei tarvitse pelätä joutuvansa kopioimaan tietohakemistossa kertaalleen olevia tietoja dokumentaatioon käsin uudestaan joka kerta, kun tiedot muuttuvat.

Onnistunut dokumentaation täydentäminen edellyttää "dokumentointivälinevetoisuutta". Tällä tarkoitan sitä, että tietohakemiston tietojen haku pitää saada kuvattua dokumentin sisään samalla välineellä, millä itse perusdokumenttikin tehdään. Hakujen sisällyttäminen ei saa turmella dokumentin varsinaisen pääosan muokkaamistyötä esimerkiksi dd. TIP-välineen tapauksessa nämä vaati-

[17]Joskus tämä saattaisi olla kaikkein parhaiten käyttäjää palvelevaa — ehkä sitomalla käsitemalliote myös käsin tehtyihin sovellusosiin niitäkin kyettäisiin monistamaan koneellisesti?

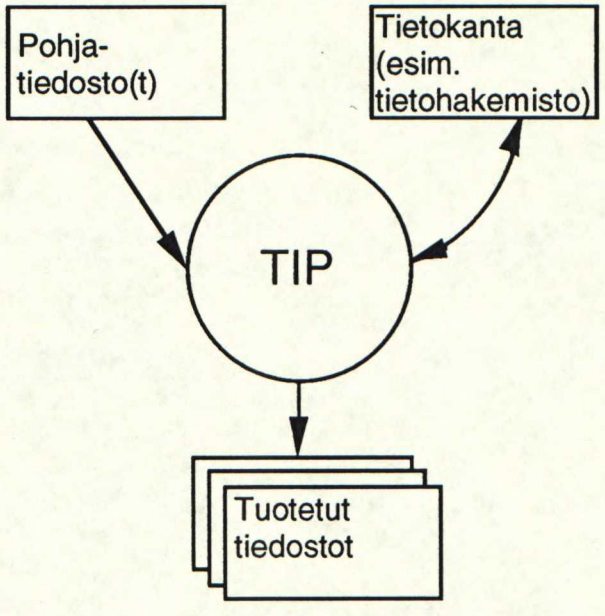
mukset toteutuvat parhaiten — ikävä kyllä — pelkkään tekstiin lomitettujen muotoilukomentojen jälkikäsittelyyn pohjaavien dokumentointijärjestelmien kanssa.

5 TIETOHAKEMISTON HYVÄKSIKÄYTTÖ TIP-TYÖKALUN AVULLA

Kohdassa 2.3, "Automaattinen koodintuotto" todettiin, että tavanomaisten sovelluskehitysvälineiden käyttöä sitovat apuvälineet ovat suhteellisen harvinaisia. Esittelen tässä jaksossa toteuttamani TIP-työkalun, jolla voidaan ohjailla tekstipohjaisten sovelluskuvausten tuottamista varsin muuntumiskykyisin tavoin.

5.1 TIP-työkalun pääperiaatteita

TIP [Mujunen1 1991] on eräänlainen tietokantaraporttigeneraattorin, makroesikääntäjän ja ohjelmoitavan tekstintoimittimen yhdistelmä. Se kykenee lukemaan tekstitiedostosta (ns. *pohjatiedosto)* toimintaohjeita ja tekstipalasia ja tuottamaan ohjeiden mukaan uusia tekstitiedostoja, joihin on yhdistelty sekä relaatiotietokannasta SQL-lauseiden avulla haettuja osia että em. pohjatiedostossa olevia tekstijaksoja.



Kuva 5–1. TIP:n toimintaympäristö.

Tarkkaan ottaen pohjatiedostossa voidaan pyytää hakemaan lisätiedostoja (tavanomainen "include") ja tietokantaa voidaan myös muuttaa SQL-lausein.

TIP:n ensimmäinen toteutus tukeutuu C-kieleen, [K&R1 1978] [K&R2 1988] ja Ingres-relaatiotiedonhallintajärjestelmän [Ingres 1990] SQL:ään VAX/VMS-ympäristössä. [Dec 1990]

Pääosa pohjatiedostossa olevasta tekstistä on tavallisesti jonkin sovellusvälineen omaa kieltä: ohjelmointikieltä, tiedonhallintajärjestelmän käyttämää DDL- tai DML-kieltä (esimerkiksi SQL) tai vaikkapa käyttöliittymävälineen lomakkeenkuvauskieltä. Tätä pohjatiedoston "varsinaista" kieltä kutsutaan isäntäkieleksi. Eräänä keskeisenä vaatimuksena TIP:lle oli, että sen tarvitsemat kieliopilliset rakenteet eivät saa häiritä isäntäkielen käyttöä juuri lainkaan. Tähän on mielestäni päästy kieliopin yksinkertaisena pitämisen avulla. TIP:n käyttämä kielioppi on esitetty liitteessä 2.

Toinen keskeinen vaatimus TIP:n toiminnalle oli kyky toteuttaa muokkauksia hyvin erityyppisille isäntäkielille, myös sellaisille, joita ei erityisesti voitu ottaa etukäteen huomioon TIP:iä suunniteltaessa. Tähän on mielestäni päästy sillä, että TIP sallii jotakuinkin mielivaltaisia tekstuaalisia käsittelyjä, samantapaisia, jotka ovat mahdollisia tavallisesti tekstintoimittimissa.

Kolmas tärkeä vaatimus oli, että TIP ei saa kiinteästi tuntea käytettävän tietokannan (tavallisesti tietohakemiston) taulujen rakennetta. Tähän päästiin sillä, että TIP sallii mielivaltaisten SQL-lauseiden käytön pohjatiedostosta käsin ja se pystyy käyttämään hyväkseen tällaisten lauseiden tuloksia tuntematta etukäteen lauseissa käsiteltävien tietojen rakennetta.

5.1.1 TIP:n käyttöalueita

TIP:iä voidaan käyttää tuottamaan tekstitiedostoja melko riippumatta siitä, mitä tiedostoilla on tarkoitus tehdä. Ensisijainen käyttö lienee tuottaa vakiomuotoista sovelluskoodia (esimerkiksi C, C++), mutta myös SQL-lauseita ja vaikkapa taulukkolaskentaohjelmille sopivia tekstitiedostoja voidaan tehdä. Seuraavassa on esitetty muutamia käyttöesimerkkejä.

- Tietohakemistossa olevien sisäkkäisten ja toisiinsa viittaavien tietotyyppimääritysten avaaminen tarpeen vaatiessa, esimerkiksi relaatiotaulun luomisessa tarvittavan rakenteettoman ("litteän") kenttäluettelon aikaansaamiseksi.
- Tietokannan tauluja vakioiduilla tavoilla käsittelevän sovelluskoodin tuottaminen tietohakemistossa olevien kenttä- ja hakuehtoluetteloiden avulla.
- Relaatiotietokannan (esimerkiksi tietohakemiston) tietojen nouto haluttuun, esim. taulukkolaskennalla jatkokäsiteltävissä olevaan tekstimuotoon. Toisin kuin pelkkä SQL, TIP pystyy purkamaan auki myös itseensä viittaavat taulut (esim. "työntekijä"-taulu, jossa on sekä esimiehiä että

alaisia ja alaisilla edelleen alaisia) sekä taulun sarakkeen arvona olevat toisten taulujen nimet.

- Jaksossa 3, "Tietohakemiston rakenteesta", esitellyn tyyppisen tietohakemiston sisällön loogisuus- ja eheystarkistusten tekeminen sekä esim. tunnuslukujen haku mielivaltaisin SQL-lausein. Tässä käytetään hyväksi TIP:n kykyä yhdistää samaan pohjatiedostoon joukko yhteenkuuluvia SQL-kyselyjä, jolloin ne tulevat tehdyksi kerralla.
- Tietokannan taulujen luonti ja jo luotujen taulujen muuttaminen. Tässä käyttötavassa tuotetaan tekstitiedostoon joukko SQL-lauseita tietohakemistossa olevien vanhojen ja sinne muutettujen uusien kenttäluetteloiden perusteella.
- Kenttien pituuksien ja tyypinmukaisten ulkoasujen haku tekstimuotoisiin käyttöliittymäkuvauksiin.
- Sovelluksen käyttöohjeiden ja dokumenttien täydentäminen ajantasalla olevilla kenttäluetteloilla, kenttien nimillä jne.

Eri käyttömahdollisuudet kuuluvat yleensä johonkin seuraavista luokista:

1) Yhdestä pohjatiedostosta monistetaan useita eri kopioita. TIP voi esim. monistaa halutun pohjatiedoston mukaisen tulostiedoston jokaista tietohakemistossa kuvattua tietokannan kohdetta (taulua) kohti. Tämä on ehkä tavoiteltavin tilanne, koska tällöin ylläpidettäviä pohjatiedostoja syntyy lukumääräisesti vähiten.

2) Yhdestä lyhyestä pohjatiedostosta tehdään yksi pitempi kopio. Esimerkiksi tietokannan taulujen luonti-SQL-lauseet tuottava pohjatiedosto on luonteeltaan tällainen.

3) Yhdestä pohjatiedostosta tehdään yksi, suunnilleen samanpituinen kopio. Näin saadaan TIP tekemään "esikäännös" ja laventamaan auki tekstiin tietokannasta (tietohakemistosta) riippuvaisia asioita. Tällaista käyttöä voisi ajatella esimerkiksi vaativan tietokantataulua käsittelevän sovelluskoodin yhteydessä: koodi sisältää paljon käsin tehtyjä osuuksia, mutta taulun rivin kentistä riippuvat jaksot saadaan kuitenkin lavennettua siihen automaattisesti TIP:n avulla. Myös dokumentaation täydentäminen tietohakemiston sisällöllä on luonteeltaan pääasiallisesti tällaista.

5.1.2 Lyhyt toiminnan kuvaus

TIP etsii pohjatiedostosta liitteessä 2 kuvattua lohkorakennetta jäsentämällä tiedoston pelkkien lohkon alku- ja loppumerkkien varassa. Tämän jälkeen TIP alkaa käydä läpi lohkorakennetta ja laventaa tulostiedostoon lohkojen tuottamia merkkijonoja.

Seuraavassa kuvassa 5–2 on esimerkki TIP:n pohjatiedostosta, joka hakee taulusta "kohteen_kentta" tiedot muistiin TIP:n symbolitauluun ja sen jälkeen käy läpi saatuja tuloksia siten, että jokaista kenttää kohti tuotetaan tekstirivi, joka on muotoa:

```
="xxx"<tab>="yyy"
```

```
="Kaikkien kohteiden kentät"
$(tip_sql kentta
    select
    * from kohteen_kentta
    order by kohteen_tunnus, lyhyt_tunnus
$)
$(tip_for_each kentta
  $(
="$(kohteen_tunnus$)"<tab>="$(lyhyt_tunnus$)"
  $)
endfor$)
$(tip_exec_sql commit work$)
```

Kuva 5–2. TIP-esimerkki.

Lohkon alussa olevaa tunnusta etsitään TIP:n symbolitaulusta. Esimerkissä lohkon tunnuksia ovat mm. "tip_sql", "tip_for_each", "kohteen_tunnus". Symbolitaulusta löytyvä lohkon tunnuksen vastine määrää, mitä ko. lohkolle tehdään.

"tip_*"-tyyppiset symbolit ovat TIP:ssä ennalta valmiiksimääriteltyjä funktioita, joilla saadaan tehtyä erilaisia ohjelmallisia operaatioita: SQL-lauseita, toistuvaa pohjatiedoston katkelman monistusta jne. Muut symbolit TIP-pohjatiedoston tekijä määrittelee eri tavoin itse symbolitauluun. Esimerkissä funktio "tip_sql" esittelee sivuvaikutuksenaan symbolin "kentta", jonka takaa löytyy suoritetun SQL-kyselyn tiedot. Varsinaisena merkkijonoarvonaan se ei palauta mitään.

Funktio "tip_for_each" puolestaan laventaa toistuvasti osan pohjatiedostoa symbolitaulusta löytyvän annetulla nimellä SQL-kyselyn lopputuloksen mukaan ja palauttaa merkkijonoarvonaan tämän lavennuksen tuloksen liitettäväksi tulostiedostoon. Kyselytuloksen jokaista riviä kohti toistetaan lohkomerkein erotettu jakso siten, että kyselyn palauttamien sarakkeiden nimet ovat suoraan käytettävissä lohkon tunnuksina. Esimerkissä näin on käytetty esimerkiksi tunnuksia "kohteen_tunnus" ja "lyhyt_tunnus". Ne palauttavat arvonaan sarakkeen arvon ko. kyselytulosrivillä merkkijonomuodossa.

Esimerkkiä tarkastelemalla on helppo nähdä TIP:n rakenteiden joustavuus ja mukautumiskysy. Esimerkkihän olisi voinut olla keskellä tekstimuotoista sovellusdokumenttia, jonne olisi tarvittu luettelo tietohakemiston kuvaamista kohteista ja niiden lyhyistä tunnuksista. Vastaavasti muokkaamalla hiukan "tip_for_each":n sisällä olevaa riviä voidaan tuottaa vaikkapa kaikki sovelluksen tietokantataulut poistava SQL-tiedosto.

TIP kykenee käyttämään symbolitauluaan assosiatiivisina taulukkoina ja epäsuorien symboliviittausten talletuspaikkana mekanismilla, joka sallii lohkon tunnuksen sisältää lavennettavia lohkoja. Toinen erittäin voimakas piirre TIP:ssä on mahdollisuus määritellä palanen TIP-lohkorakennetta symbolitaulussa olevan tunnuksen arvoksi ja myöhemmin käyttää tätä tunnusta aliohjelman/makron tapaan toisen lohkon tunnuksena. Tällaiselle TIP-kieliselle (lavennusaikaiselle) aliohjelmalle voidaan antaa myös eri käyttöpaikoissa erilaisia parametreja. Näitä ominaisuuksia havainnollistetaan seuraavan kuvan 5–3 pohjatiedostossa.

```
$(tip_declare klo $(tip_$(1$)$)$)abc
$(klo time$)
def
$(klo date$)
xyz
$(klo time$)
```

Kuva 5–3. Toinen TIP-esimerkki.

ja tulokseksi saadaan kuvan 5–4 tulostiedosto (jos oletetaan, että koneessa on niin paljon kuormaa, että TIP toimii todella hitaasti):

```
abc
13:21:03
def
12.03.1991
xyz
13:21:05
```

Kuva 5–4. Toisen TIP-esimerkin tulos.

Todellisista TIP:n käyttötilanteista peräisin olevia TIP-pohja- ja tulostiedostoja on otettu mukaan liitteeseen 3 esimerkeiksi TIP:n käyttömahdollisuuksista ja hieman todentuntuisemmista pohjatiedostoista.

Tarkka kuvaus TIP:n toiminnasta ja käytettävistä olevista valmiista funktioista on lähteessä [Mujunen2 1991].

Jakson lopuissa kohdissa tutustutaan TIP:n soveltamismahdollisuuksiin erilaisissa jaksoissa 3 ja 4 esitellyissä tilanteissa.

5.2 Käsitemalli ja tietokannan rakenne

Kuten jaksossa 3, "Tietohakemiston rakenteesta" todettiin, tietohakemistoon talletettavan käsitemallin odotetaan esittävän todellista tietokantatoteutusta. Eräs kiintoisa mahdollisuus soveltaa TIP:ä tässä yhteydessä voisi olla jollain CASE-välineellä piirretyn käsitekaavion sisältämän informaation vienti tietohakemistoon.

Esimerkiksi Prosa-työkalun (ks. kohta 3.2.3.1, "Insoft Prosa, Cadre Teamwork") tuottaman, kaavion faktat sisältävän tekstimuotoisen tiedoston voisi melko yksinkertaisella apuohjelmalla muuntaa TIP-rakenteita sisältäväksi TIP:n syöttötiedostoksi. Tämän tiedoston käsittelyn tuloksena kaavion yksilötyypit ja yhteydet vietäisiin yhteisessä pohjatiedostossa olevin TIP-kielisin SQL:ää käyttävin aliohjelmin tietohakemistoon ja sivutuotteena syntyisi raportti tehdyistä tietohakemistomuutoksista.

5.3 Käsitemallin ja tietokannan muutokset

Käsitemallin yksilötyyppi- ja yhteystasoiset muutokset viedään tietohakemiston käsitemallia kuvaavaan osaan ("Kohde", "Yhteys") joko tietohakemiston ylläpito-

ohjelmin tai edellisessä kohdassa 5.2 kuvatun kaaviosta johtamisen avulla. Käsitteiden tietokuvausten muutokset päivitetään puolestaan tietohakemiston tyyppijärjestelmään, jonka jälkeen voidaan tuottaa TIP:ä käyttäen uusi versio kohteiden kentistä.

Kohteen tietosisällön kuvaavan (ryhmä-)tyypin kuvaus palautetaan pelkästään perustyyppejä sisältäväksi alkeiskenttien luetteloksi TIP-pohjatiedostolla, jossa haetaan tietohakemistosta joukko kohteita, joista jokaista kohti kutsutaan TIP-kielistä aliohjelmaa, jonka tehtävänä on purkaa tietohakemistossa esiintyvä tyyppi.

Mikäli tyyppi on lajiltaan ryhmä- tai yksinkertainen tyyppi, ko. aliohjelma kutsuu itseään purkamaan tyypin määrittelyyn kuuluvat tyypit (tyypin) ja välittää tällä tasolla olevat tyypin parametrit sekä toteutuskuvaukset yhdistettäviksi sisemmillä tasoilla vastaan tuleviin.

Mikäli tyyppi on lajiltaan perustyyppi, aliohjelma kirjaa sen käsittelyssä olevan kohteen kentäksi tietohakemistoon täydennettynä rekursiossa kootuilla tyypin parametreilla ja toteutuskuvauksilla.

Liitteessä 3 on edellä kuvattu sisäkkäisyyden purkava osa tyyppijärjestelmän läpikäyntiin käytetystä "tihlit.tip"-pohjatiedostosta. Toisena tähän kohtaan liittyvänä esimerkkinä on uusien taulujen pelkät luonti-SQL-lauseet tuottava pohjatiedosto "ct.tip" kokonaisuudessaan.

Uutta kenttäluetteloa talletettaessa käytetään uutta versiotunnusta. Näin "Kohteen kenttä"-tauluun jää jäljelle myös ennen tietokuvausten muutoksia vallinnut (mahdollisesti relaatiokannaksi asti toteutettu) tilanne.

Näitä "uutta" ja "vanhaa" kenttäluetteloa vertailemalla pystytään lähes automaattisesti tuottamaan vanhat tiedot uuteen taulurakenteeseen muuntava SQL-lausejoukko. Tämä on erityisen välttämätöntä siksi, että standardi-SQL:ssä ei tarvitse toteuttaa mitään taulujen muutoslauseita ("ALTER TABLE"). Ainoa varmasti kaikissa SQL-järjestelmissä toimiva menettelytapa muuttaa tauluja on:

1) Talletetaan vanhat tiedot turvaan (ulkoiseen tiedostoon tai aputauluun).
2) Poistetaan vanha taulu.
3) Luodaan uusi taulu uusin kentin.
4) Täytetään uusi taulu vanhoilla tiedoilla.

Mikäli käytettävässä relaatiojärjestelmässä on käytettävissä taulun nimen muutos, kannattaa kohdat 1 ja 2 korvata pelkällä vanhan taulun nimen muutoksella. Vanhat kenttäluettelot voidaan versiotunnuksen turvin jättää talteen dokumentoimaan joskus vallinnut tietokannan rakenne.

Kohde- ja kenttämuutokset voidaan luokitella esimerkiksi seuraaviin ryhmiin.

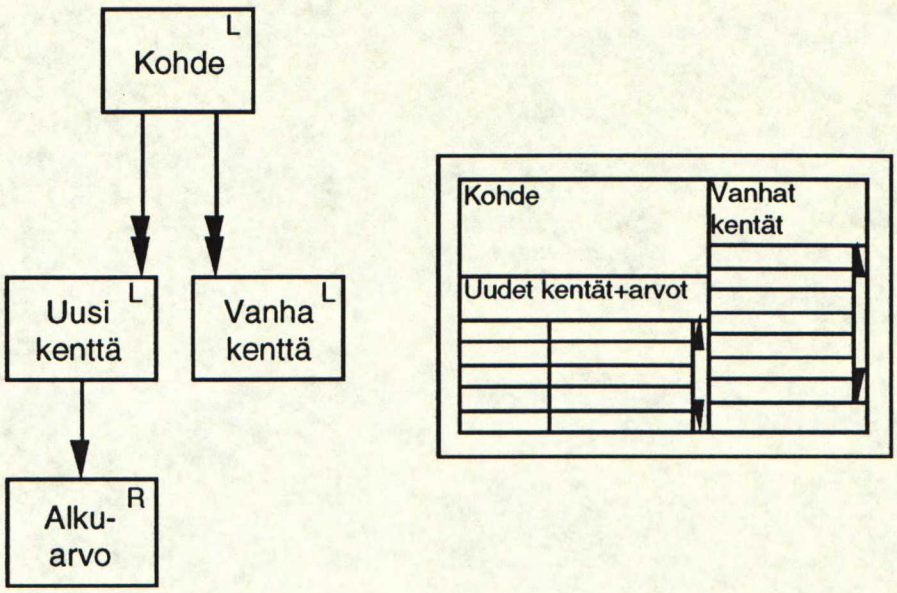
Tavallisimmat tietomuutokset ovat pelkkien yksittäisten perustietotyyppitasoisten kenttien pidennys, lyhennys tai tyyppimuutos. Ongelmia voi seurata yrityksestä tehdä tietoa hävittäviä muutoksia kuten kentän lyhennys tai tyyppimuunnos (esimerkiksi liukuluvusta kokonaisluvuksi). Näistä selvitään useimmiten kertomalla kohde-SQL-järjestelmälle eksplisiittisillä muunnosfunktioilla, että tietoa saa kadota.

Uudet kentät tunnistetaan siitä, että niitä ei ole vanhassa kenttäluettelossa. Lisättäessä uusia kenttiä niille tarvitaan oletusarvot, jollaisina on helpointa käyttää joko vakiolausekkeita tai vanhan taulun sarakkeisiin nojaavia SQL-lausekkeita. Näitä molempia voidaan sijoittaa taulumuutoksen vaiheesta 4, uuden taulun täyttö, tuotettavaan "INSERT...AS SELECT..."-SQL-lauseen SELECT-osaan.

Vanhojen kenttien poiston tunnistaa siitä, että niitä ei ole enää uudessa kenttäluettelossa. Poistettaviin kenttiin talletetut tiedot voidaan joko hävittää, siirtää tilapäiseen aputauluun tai uuden taulun toisennimisen sarakkeen arvoksi. Viimeksimainittu tapaus tarkoittaa itse asiassa sarakkeen nimen vaihtoa.

Käsin tehtäväksi työksi jää siis keksiä uusille kentille sopivat oletusarvot ja huomata tässä yhteydessä, mitkä kentät ovat vain saaneet uuden nimen ja jotka näin saavat oletusarvoikseen vanhannimisessä sarakkeessa olleet arvot. Poistuvista kentistä tulee tarkistaa, että joko niiden sisältämät tiedot ovat tarpeettomat tai vaihtoehtoisesti niiden arvot on käytetty hyväksi rakennettaessa joillekin uusille sarakkeille oletusarvoja.

Tämä käsin tehtävä tarkistus vaatii käytännössä sitä varten tarkoitetun lomakemuotoisen ohjelman, johon saadaan näkyviin uusien taulujen rakenne ja johon on jokaisen uuden kentän viereen varattu tilaa ko. kentän oletusarvon lausekkeelle. Samannimiset vanhat kentät on siirretty vastaaville paikoilleen oletuksen mukaisiksi oletusarvoiksi. Loput vanhat kentät ovat omassa luettelossaan, josta niitä voidaan ottaa käyttöön tai hylätä. Lomakkeen käsitemalliotekuvaus on kuvassa 5–5.



Kuva 5–5. Oletusarvot uusille kentille.

Vastaavanlainen järjestely voidaan tehdä myös kohteille/tauluille. Uusille tauluille on vain selvästi vaikeampaa kuvailla muunlaista oletusarvoa kuin joko taulun jättäminen tyhjäksi tai sen täyttäminen jonkin vanhan taulun kaikilla tiedoilla.

Muutos-SQL-lauseiden tuottamisen jälkeen ne suoritetaan tarvituissa vanhaa tietokantaa käyttävissä installaatioissa.

Tässä kohdassa kuvatun automatisoidun taulumuutosmenettelyn suurimpana vaikeutena käytännössä on taulujen kopioinnin vaatima levytila ja aika. Hiukankin suurempikokoisten taulujen yhteydessä joudutaan kertomaan apu- ja uusien taulujen luonti-SQL-lauseissa, minne taulut on sijoitettava. Aputaulujen kohdistaminen automaattisesti kohti vapaata levytilaa ei ole kovin yksinkertaista, varsinkin, kun taulun tietylle levylaitteelle sijoittamismenettely vaihtelee laajasti eri SQL-toteutuksissa.

5.4 Tietokannan rakenteeseen liittyvät sovellusosat

Tietokannan rakenteen muututtua ensimmäinen muutoksiin mukautumaan joutuva osa sovelluksesta on lähellä tiedonhallintajärjestelmää toimiva ja SQL-lauseita sisältävä ohjelmiston osa. Tämä osa voi olla vaikkapa joukko jollain olio-suuntautuneella ohjelmointikielellä toteutettuja luokkia, jotka kykenevät suorittamaan perusoperaatioihin "lisäys", "poisto", "muutos", "haku" liittyviä SQL-lauseita.

Tietokantataulun kenttäluetteloa voidaan TIP:n avulla monistaa ainakin tarvittujen muistipuskureiden kuvauksiin sekä tietenkin SQL-lauseiden sisälle. Puskureiden kuvauksista on esimerkkipohja "koyhth.tip" liitteessä 3.

5.5 Käyttöliittymän rakenteeseen liittyvät sovellusosat

Yksittäisten tietokenttien "pienet" pituus- ja tyyppimuutokset eivät vaadi käsityötä olemassaolevan lomakkeen parissa, mikäli käytetty käyttöliittymäväline pystyy sopeutumaan kenttien muutoksiin ilman ihmisen tekemää hienosäätöä. Sopeutuminen tarkoittaa sitä, että väline pystyy mitoittamaan lomakkeen uudelleen, mikäli tehdyt muutokset ovat niin pieniä, että uudelleenmitoittaminen on edes periaatteessa mahdollista. Periaatteessa mahdottomaksi se muuttuu esimerkiksi silloin, kun vierekkäin aseteltaviksi määrätyt kentät eivät yksinkertaisesti mahdu kuvaruudun tai ikkunan mittoihin. Joka tapauksessa lienee mielekästä tarkistaa tuottamisen jälkeen, miltä koneellisesti uudelleen tuotetut lomakkeet näyttävät.

TIP voi sopivan pohjatiedoston avulla etsiä tietohakemiston tyyppijärjestelmästä kaikki ne tyypit, joihin tietty tyyppimuutos vaikuttaa. Tyypeistä voidaan edetä etsimään muuttuvat kohteet ja kohteista kohdepalat. Välittömästi tyyppimuutos vaikuttaa juuri kohdepalojen ulkoasuihin, ja mikäli käyttöliittymätyökalu on riittävän kehittynyt korjaamaan vain lomakkeiden osia, saatetaan selvitä pelkkien kohdepalojen korjaamisella. Mikäli halutaan tarkistaa kaikki muutoksia epäsuorastikin kokeneet lomakkeet, joudutaan etenemään kohdepaloista vielä kaikkiin sellaisiin pääkohteina toimiviin lomakepaloihin, joiden osana on jokin muuttunut alipala. Tällä tavoin saatu lomakeluettelo on tarkistuslistana läpikäytäessä muuttuneita lomakkeita. Saman listan avulla voidaan korjata käsin lomakkeet, joita jokin laajempi tyyppijärjestelmän muutos koskee.

Käyttöliittymän käsin tapahtuvan "hienosäädön" tallettamista johonkin kuvaukseen sellaisessa muodossa, joka taltioisi juuri tehdyn hienosäätömuutoksen eikä säädön lopputuloksena olevaa kenttää, kannattaisi yrittää kehittää. Esimerkkeinä asioista, jonka kaltaisia voisi yrittää tallettaa, ovat esimerkiksi "lyhennä tätä kenttää niin, että sen jälkeen tuleva kenttä mahtuu" tai "jos seliteteksti on yli 25% pidempi kuin minkään toisen samassa pystypalstassa olevan kentän seliteteksti, katkaise selite". Nämä ovat esimerkkeinä perusteista, jollaisten varassa ihminen muokkaa lomakkeita visuaalisesti miellyttävään muotoon.

Esimerkistä voisi kuvitella, että etsin käyttöliittymien tuottamiseen jotain tietämyspohjaista asiantuntijajärjestelmää, johon on kerätty kattavat ja kaikkiin

tilanteisiin sopivat säännöt lomakkeiden laatimiseksi. Tavoittelen kuitenkin astetta vaatimattomampaa taltiointijärjestelmää. Haluaisin löytää menettelyitä, joilla lomakkeen laatija voisi ensimmäistä kertaa tiettyä lomaketta toteuttaessaan saada talteen ko. lomakkeen suunnitteluun vaikuttaneita päätöksiä ja sääntöjä. Näitä olisi tarkoitus soveltaa uudelleen vain tuohon samaan lomakkeeseen silloin, kun sen pohjana olevien kohteiden tietosisällöt muuttuvat.

5.6 Ohjelmointikielten puutteiden kompensointi

TIP:ä voidaan käyttää myös ilman liittymää tietohakemistoon[18] kompensoimaan ohjelmointikielten uudelleenkäytettävyyspiirteiden puutteita. TIP:llä voi toteuttaa kieleen toisten tiedostojen sisällyttämiset ja parametroidut makrolavennukset hieman C-kielen esikääntäjää vastaavaan tapaan.[19] Myös ajon aikana kutsuttavien aliohjelmien puutetta voi korvata jossain määrin käyttämällä makroja aukilavennettavien aliohjelmien (engl. *inline subprograms)* tapaan.

Parhaimmillaan TIP on kuitenkin kompensoidessaan sellaisia useista ohjelmointikielistä puuttuvia ominaisuuksia kuten aliohjelman mukautuminen käsiteltävän tiedon tyyppiin (polymorfismi, engl. *polymorphism)* tai lausejonon toistaminen tietuetyypin jokaiselle kentälle. Staattista (TIP-aikaista) polymorfismia TIP voi simuloida laventamalla lopputulokseen tietotyyppien nimistä riippuvaisille tunnuksille TIP:n symbolitauluun sijoitettuja koodijaksoja. Tietuetyypin kentät voidaan TIP:lle kuvata määrittelemällä ne pohjatiedostossa symbolitauluun taulukkomuodossa ja käyttämällä sitten TIP:n toistofunktioita tuottamaan sekä varsinaiset tietuetyypin määritykset että tietueen kentistä riippuvaiset koodijaksot. Tällaista "tiedostoon kuvattua tietohakemistoa" voidaan käyttää hyvin täydentämässä relaatiotietokannassa olevia tietohakemistotauluja, mikäli useamman yhtäaikaisen käyttäjän ei tarvitse muokata tiedostossa olevia kuvauksia.

TIP pystyy tallettamaan pysyväksi ja uudelleensuoritettavaksi kuvaukseksi monet sellaiset asiat, jotka aiemmin voitiin tallettaa vain tekstitiedoston muokkausohjeiksi, kuten esimerkiksi ohjelmajaksojen koostamisen mielivaltaisista tekstifragmenteista, joiden sisällä tarvitaan vielä vaihtelevia tekstikorvauksia.

---

[18]Tietenkin tässä kohdassa esitettyjä piirteitä käytetään myös tietohakemiston kanssa; nämä piirteet vain ovat hyödyllisiä myös ilman sitä.

[19]Käytön kannalta TIP:n makrot eivät aseta lähdetekstissä sallituille makron käyttöpaikoille mitään rajoituksia toisin kuin C-esikääntäjä, joka käsittää makrojen käytöksi vain C:n syntaksin mukaiset tunnukset. TIP sen sijaan sallii makrolavennukset jopa makron nimessä.

6 JOHTOPÄÄTÖKSET

Tässä työssä on tutustuttu mahdollisuuksiin yhdistää sopivalla työkalulla käsitemallipohjainen tietohakemisto ja perinteiset sovelluskehitysvälineet kaupallis-hallinnollisten sovellusten automaattisempaan tuotantoon ja ylläpitoon.

Eräs tärkeä havainto työn aihepiiriin tutustuttaessa oli, että nykyään kaupallis-hallinnollisia sovelluksia ylläpidettäessä tehdään paljon hidasta käsityötä — juuri sellaista työtä, joka olemukseltaan on yhden tietyn muutoksen käsin tapahtuvaa levittämistä kaikkiin käyttöpaikkoihin, joihin se on (käsin) monistettu sovellusta alunperin tehtäessä. Hyvin harvoin käsin monistamisen sijaan taltioidaan pysyvästi tietoa siitä, millainen monistus pitää tehdä, ja annetaan jonkin apuvälineen huolehtia monistuksesta.

Työssä kuvatusta tietohakemistosta on Tietosavo Oy:ssä toteutettu prototyyppi. Sen käyttökelpoisuutta ja tietohakemistoratkaisun mallinnuskykyä on testattu kokeilujen ohella tallettamalla siihen erään Ingres-relaatiotietokannan ja 4GL-kielen varaan rakennetun sovellusosan käsitemallin ja tietokantaratkaisun osia ja todettu, että esimerkiksi yksinkertaisiin relaatiotaulujen luontitehtäviin ratkaisu on erittäin toimiva. Monimutkaisempiin sovelluskoodia sisältäviin TIP-pohja-tiedostoihin täytyy vielä uhrata lisätyötä ennen kuin voidaan luotettavasti arvioida todellista työn tehostumisen astetta. Lisäksi käyttöliittymän (merkkipohjaisten ja graafisten lomakkeiden) kuvaamiseen tällä hetkellä Tietosavo Oy:ssä käytetyt välineet eivät kykene ilman merkittäviä itsetehtyjä lisäapuvälineitä käyttämään syötteenään tekstimuotoisia kuvauksia, joten tietohakemistopohjaista lomakkeiden tuottamista ei ole voitu käytännössä vielä kokeilla.

Tietohakemiston edelleenkehittämisen painopisteitä ovat tietokantataulujen yhteydessä käytettyjen hakuehtojen sekä taulujen pää- ja vierasavainkenttien kuvaaminen. Näin saataisiin automatisoitua lisää relaatiotietokannoissa tarvittavien hakuindeksien luontia ja lomakkeiden välisten siirtymisten tuottamista.

Ehkä ilahduttavinta työn tekijän kannalta on se, että työn osana toteutettu TIP-työkalu on ollut heinäkuusta 1991 lähtien jatkuvassa käytössä Tietosavo Oy:ssä tuotettaessa Ingres-SQL-, Ingres-4GL-, Microsoft Visual Basic- ja C-kielisiä sovellusosia. Ainakin riippumattomuus käytettävästä sovellusvälineestä näyttäisi toteutuneen, sillä TIP:n suunnittelun aikaan ei aavistettu, että sillä joskus tuotettaisiin Ingres-4GL:ää ja Visual Basic -välinettä ei tuolloin ollut edes markkinoilla. TIP:n voidaan siis katsoa saavuttaneen ainakin osan kehitystyössä sille asetetuista tavoitteista.

TIP-välinettä kannattaisi kehittää käyttäjän kannalta ystävällisempiin suuntiin. Sen virhediagnostiikassa, TIP-aliohjelmien parametrinvälityksessä ja muistinkäytössä on kehittämisen varaa. Käytännössä TIP:n yksinkertainen ja isäntäkieliä häiritsemätön kielioppi on jonkin verran ikävää kirjoitettavaa mm. tarkkojen lohkorakenteidensa takia. Saattaisi olla mahdollista kehittää TIP:lle rinnakkainen kielioppi, joka olisi periaatteeltaan käännetty juuri päinvastoin: isäntäkieliset jaksot jouduttaisiin merkitsemään erikoismerkein, mutta TIP:n omat lohkorakenteet voitaisiin kirjoittaa muistuttamaan enemmän perinteisiä ohjelmointikieliä. Olisi erinomaista, mikäli semanttisesti saman kielen molemmat syntaktiset ilmiasut voitaisiin saada samaan TIP-toteutukseen, jotta käyttäjä voisi valita niiden välillä kulloisenkin käyttötilanteen mukaan.

LÄHDELUETTELO

[Ada 1983] ANSI/MIL-STD-1815A-1983. Reference Manual for the Ada Programming Language. American National Standards Institute Inc., United States Department of Defence, Washington, USA. Useita julkaisijoita, mm: Rogers, M. W. (toim.) 1984. Ada: Language, compilers and bibliography. 1. p. Cambridge, Cambridge University Press. (Epäjatkuva sivunumerointi, n. 500 s.)

[Backus 1981] Backus, John 1981. History of FORTRAN I, II, and III. Teoksessa Wexelblatt R. L. (toim.) 1981. History of Programming Languages. Academic Press Inc.

[Borland 1990] Borland International Inc. 1990. Turbo Pascal Version 6.0: TurboVision Guide. Scotts Valley, California, USA, Borland International Inc. 409 s.

[Cadre 1988] Cadre Technologies Inc. 1988. Teamwork Management Overview. Providence, RI, USA, Cadre Technologies Inc. 28 s.

[Chen 1976] Chen, Peter 1976. The Entity-Relationship Model: Toward a Unified View of Data. ACM Transactions on Database Systems, Vol 1, No 1, March 1976, ss. 9–36.

[Chen 1981] Chen, Peter (toim.) 1981. Entity-Relationship Approach to Information Modeling and Analysis. Proceedings of the Second International Conference on Entity-Relationship Approach, Washington, D.C., October 12-14, 1981. Saugus, CA, USA, ER Institute; Amsterdam, Netherlands, Elsevier Science Publishers B.V. 602 s.

[Ceri 1991] Ceri S., Tanca L., Zicari R 1991. Supporting Interoperability between New Database Languages. Dipartimento di Elettronica, Politecnico di Milano, Piazza L. da Vinci 32, 20133 Milano (ceri@ipmel1.polimi.it).

[Coad 1991] Coad, Peter & Yourdon, Edward 1991. Object-Oriented Analysis. 2. p. New Jersey, Yourdon Press, a Prentice-Hall Company, a Division of Simon & Schuster. 233 s.

[Dec 1990] Digital Equipment Corporation 1990. VMS Extended Documentation Set. Tilausnumero QA-001AA-GZ.5.4. Maynard, Massachusetts, USA, Digital Equipment Corporation. Noin 40 kirjaa/mappia.

[Deft 1988] Deft User's Guide. Toronto, DEFT Inc., 1988.

[DeMarco 1979] DeMarco, Tom 1979. Structured Analysis and System Specification. 20. p. New Jersey, Yourdon Press, a Prentice-Hall Company, a Division of Simon & Schuster. 352 s.

[Excelerator 1989] Index Technology Corporation 1989. Excelerator/IS, Facilities & Functions Reference Guide, Release 1.9. Cambridge, Massachusetts, USA, Index Technology Corporation. Yritys on nykyisin nimeltään "Intersolv".

[Friedman 1991] Friedman, Linda Weiser 1991. Comparative Programming Languages — Generalizing the Programming Function. New Jersey, Prentice-Hall Inc. 578 s.

[IBM1 1991] IBM Announcement Letter #ZP91-05545, Sept. 1991. Repository Manager/MVS Version 1 Release 2 Modification 1. IBM Europe, Middle East and Africa. 8 s.

[IBM2 1991] International Business Machines Corporation 1991. Systems Application Acrhitecture, AD/Cycle Concepts. 2. p. Tilausnumero GC26-4531-01. Armonk, New York, USA, International Business Machines Corporation. 28 s.

[IBM3 1990] International Business Machines Corporation 1990. Repository Manager/MVS, General Information, Version 1 Release 2. 2. p. Tilausnumero GC26-4608-1. Armonk, New York, USA, International Business Machines Corporation. 47 s.

[IBM4 1991] International Business Machines Corporation 1991. Repository Manager/MVS, Guide for Users and Administrators, Release 2. 2. p. Tilausnumero SC26-4612-01. Armonk, New York, USA, International Business Machines Corporation. 312 s.

[Ingres 1990] Relational Technology Inc. 1990. Ingres/SQL Reference. Alameda, California, USA, Relational Technology Inc. (Epäjatkuva sivunumerointi, n. 600 s.)

[K&R1 1978] Kernighan, Brian W. & Ritchie, Dennis M. 1978. C Programming Language. 1. p. New Jersey, Prentice-Hall Software Series. 228 s.

[K&R2 1988] Kernighan, Brian W. & Ritchie, Dennis M. 1988. C Programming Language. 2. p. New Jersey, Prentice-Hall Software Series. 272 s.

[Kim 1989] Kim, Won & Lochovsky, Frederick H. (toim.) 1989. Object-Oriented Concepts, Databases, and Applications. 2. p. New York, ACM Press, an Addison-Wesley Publishing Company. 602 s.

[Liskov 1981] Liskov, Barbara et al 1981. The Programming Language CLU. 1. p. Berlin-Heidelberg-New York, Springer Verlag. 190 s.

[Martin 1976] Martin, James 1976. Principles of Data-Base Management. New Jersey, Prentice-Hall Inc.

[Martin 1982] Martin, James 1982. Application Development Without Programmers. 3. p New Jersey, Prentice-Hall Inc. 350 s.

[Meyer 1988] Meyer, Bertrand 1988. Object-Oriented Software Construction. 1. p. Hertfordshire, Prentice-Hall International (UK) Ltd, a Division of Simon&Schuster. 534 s.

[Mujunen1 1991] Mujunen, Ari 1991. TIP-käyttöohjeet. Kuopio, Tietosavo Oy. 59 s. Tilausosoite: Tietosavo Oy, Mikko Kolehmainen, PL 1582, 70461 KUOPIO.

[Mujunen2 1991] Mujunen, Ari 1991. Oliosuuntautuneiden tekniikoiden soveltamismahdollisuuksia kaupallisesti saatavilla olevien relaatiotietokantojen yhteydessä. Erikoistyö, Teknillinen Korkeakoulu, tietotekniikan osasto, tietojenkäsittelyopin laboratorio. 25 s.

[Prosa 1989] prosa STRUCTURED ANALYSIS. User's Manual v3.00. Oulu, Insoft Ky, 1989.

[SFS 1988] SFS-käsikirja 106, Systeemityön menetelmä. ERTI-käsiteanalyysi. Suomen Standardisoimisliitto, 1988.

[Turtiainen 1990] Turtiainen, Pasi 1990. Kokemuksia SA-menetelmän käytöstä sulautetun mikroprosessoriohjelmiston suunnittelussa. Diplomityö, Teknillinen korkeakoulu, sähkötekniikan osasto. 72 s.

[Ullman 1988] Ullman, Jeffrey P. 1988. Principles of Database and Knowledge-based Systems. Vol. 1. 1.p. Rockville, MD, USA, Computer Science Press. 631 s.

[Wirth 1988] Wirth, Niklaus 1988. From Modula to Oberon. Software — Practise and Experience, Vol 18, No 7, July 1988.

[Yourdon 1989] Yourdon, Edward 1989. Modern Structured Analysis. New Jersey, Yourdon Press, a Prentice-Hall Company, a Division of Simon & Schuster.

Työn tekemisen välineistä

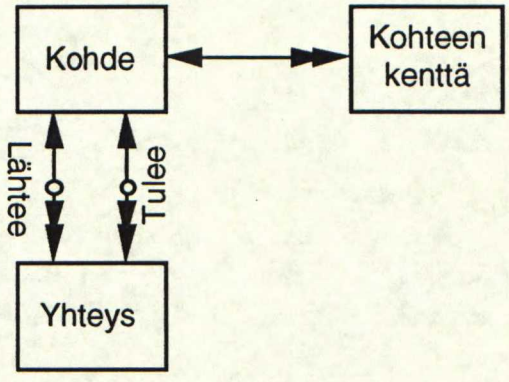
Tämä työ on tuotettu kokonaisuudessaan käyttäen apuna Apple Macintosh IIci -tietokonetta ja "WriteNow 2.0"-tekstinkäsittelyohjelmistoa. Kuvitukset on tehty "MacDraw II v1.1" -ohjelmalla ja sijoitettu koneluettavassa muodossa tekstin joukkoon. Originaalivedos on tulostettu "QMS-PS 410"-PostScript-laserkirjoittimella. Leipätekstissä on käytetty 12 pisteen "Adobe Garamond"-kirjasinta ja otsikoinnissa 12 pisteen "Helvetica" -kirjasinta. Alaviitteiden kirjasimena on 10 pisteen "Adobe Garamond", erilaisissa ruutuvedoksissa ja tietokonetulostenäytteissä 9, 10 tai 12 pisteen "Courier" ja kuvien sisällä joko 10 tai 12 pisteen "Helvetica".

Tässä yhteydessä haluan osoittaa kiitokseni kaikille niille lahjakkaille henkilöille Apple Computer Inc. -yhtiössä, jotka ovat kyenneet luomaan sellaisia henkilökohtaisia tietokoneita, joiden käyttämisestä ei tarvitse tulla päätoimisen opiskelun kohdetta. Ilman tällaista apuvälinettä ylimääräistä energiaa olisi kulunut hukkaan työn sisällön kannalta toisarvoisista seikoista huolehtimiseen.

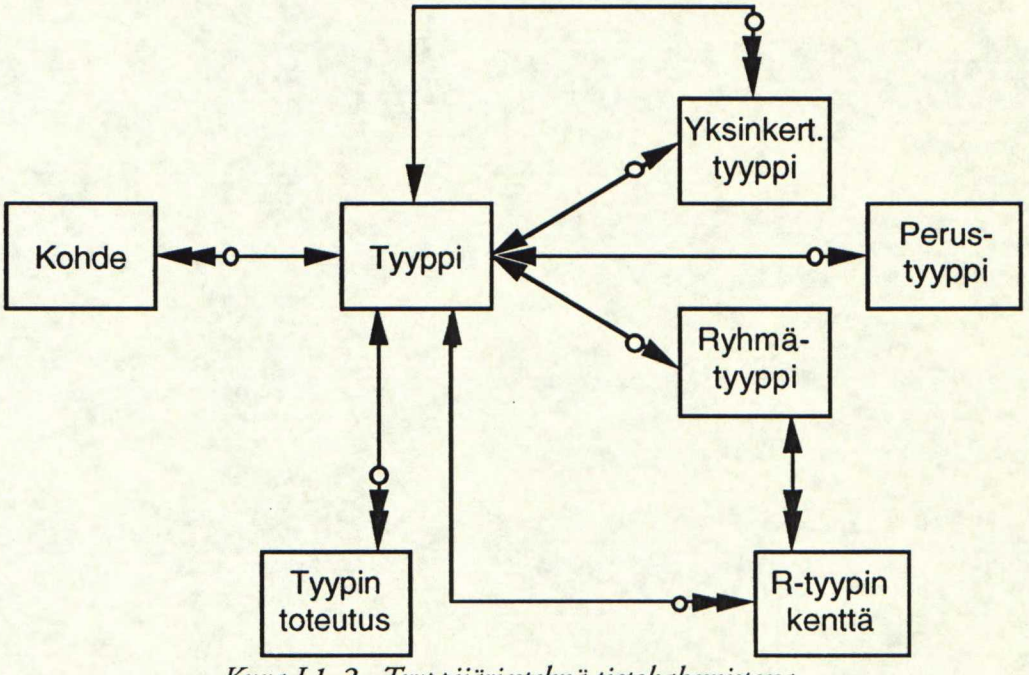
LIITTEET

Liite 1. Laadittu tietohakemiston käsitemalli

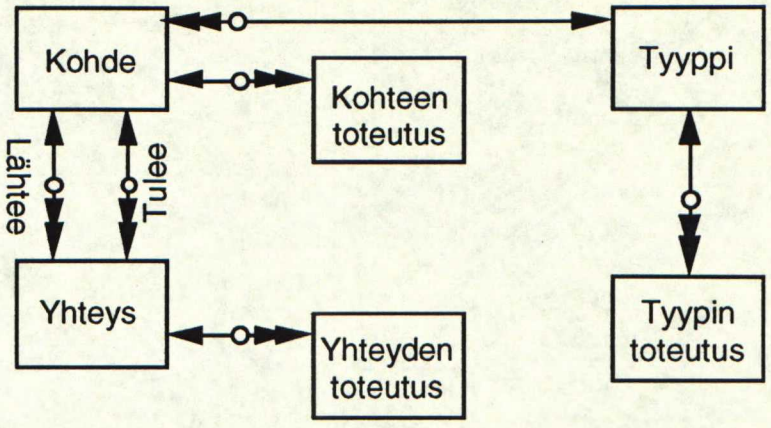
Tähän liitteeseen on koottu työssä osina esitellyn tietohakemiston käsitekaaviot yhdeksi kokonaisuudeksi.



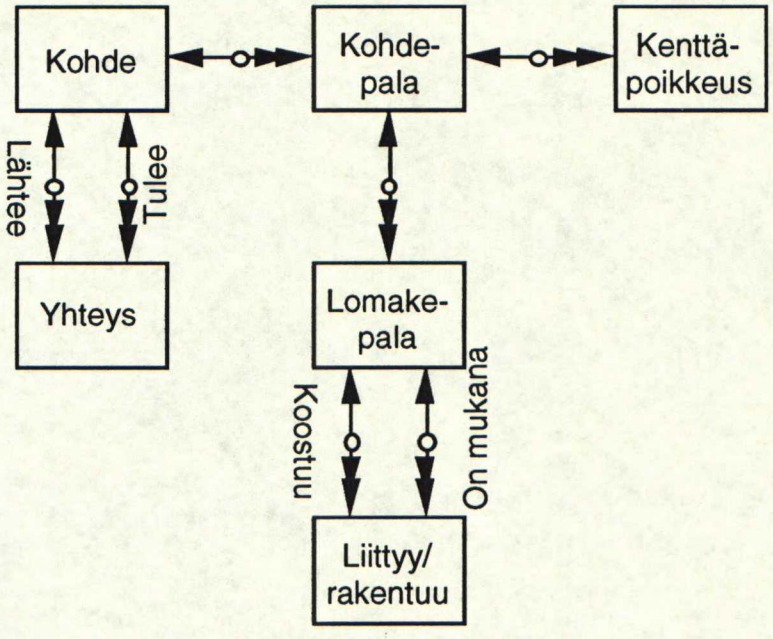
Kuva L1–1. Käsitemalli tietohakemistossa.



Kuva L1–2. Tyyppijärjestelmä tietohakemistossa.



Kuva L1–3. Toteutuskuvaukset tietohakemistossa.



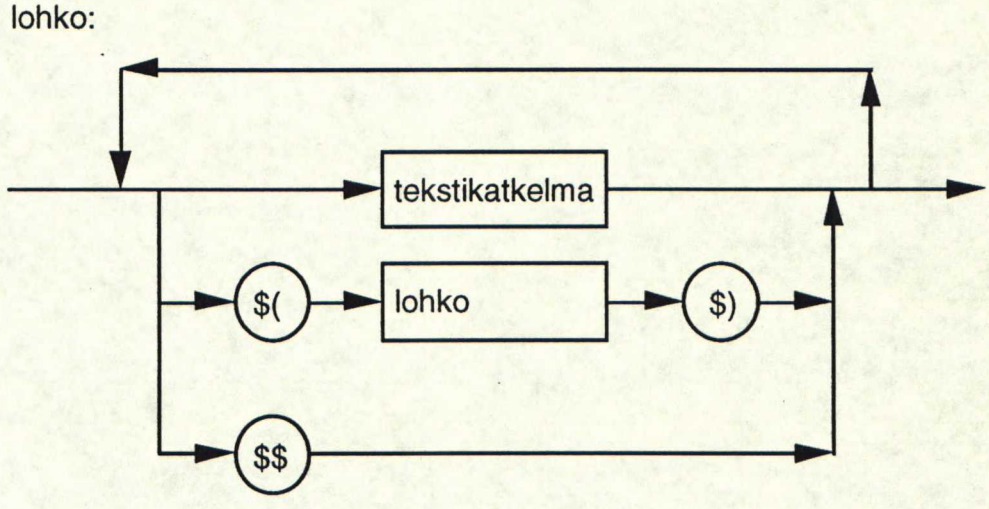
Kuva L1–4. Käsitemalliotteet tietohakemistossa.

Liite 2. TIP-työkalun käsittelemän kielen kielioppi

Koska TIP:n suunnittelun keskeisenä tavoitteena oli pystyä lomittumaan jonkin toisen, ns. isäntäkielen joukkoon, se ei voinut "varata" itselleen esimerkiksi suurta joukkoa erikoismerkkejä tms. Tällainen olisi vaarantanut TIP:n käyttökelpoisuuden sellaisten tekstiesitysten yhteydessä, joissa TIP:n tarvitsemat merkinnät ovat jo käytössä. Näin päädyttiin ratkaisuun, jossa TIP tunnistaa vain yhtä erikoismerkkiä, joka on vielä tarvittaessa helposti vaihdettavissa toiseksi, mikäli käytettävän isäntäkielen kannalta se on edullista.

Toisaalta TIP-rakenteilta etsittiin mahdollisimman suurta joustavuutta ja ilmaisukykyä. Rajattomasti sisäkkäisyyttä salliva lohkorakenne vaikutti tarkoitukseen sopivalta.

Seuraavassa kuvassa L2–1 on esitetty syntaksidiagrammin muodossa TIP-työkalun syöttötekstitiedoston kieliopillinen rakenne.



Kuva L2–1. TIP-kielioppikaavio.

Kaaviossa esiintyvä "tekstikatkelma" määritellään sellaisena yhtenäisenä (ASCII-merkkejä) sisältävänä merkkijonona, jossa ei ole alimerkkijonoina "$(", "$)" eikä "$$".[20] Terminaali "$$" muunnetaan tekstikatkelmaksi, joka koostuu yhdestä merkistä "$". (Tämän tarkoituksena on sallia "$"-merkkien kirjoittaminen tekstikatkelmiksi.)

Tälle yksinkertaisen syntaksin perustalle rakentuu myös TIP:n käsittelemän kielen perussemantiikka. Kokonainen TIP-lähdetekstitiedosto tulkitaan yhdeksi lohkoksi, jonka arvona syntyvä merkkijono kirjoitetaan tulostekstitiedostoon.

[20] Itse asiassa kaikki merkkiparit, joiden ensimmäisenä merkkinä on "$", on varattu tulevia kielen laajennuksia varten.

Lohkon arvomerkkijono rakennetaan yhdistämällä siinä peräkkäin olevien tekstikatkelmien, alilohkojen ja "$"-tekstikatkelmien synnyttämät merkkijonot peräkkäin. Tekstikatkelma synnyttää arvonaan itsensä. Alilohko laventuu jonkin muun säännön mukaan (mahdollisesti tyhjäksi, ei merkkejä sisältäväksi) merkkijonoksi. "$"-tekstikatkelma laventuu merkkijonoksi, jossa on yksi merkki, "$".

TIP:n käsittelemän kielen joustavuus perustuu pääosin siihen vapauteen, mitä voidaan käyttää päätettäessä, kuinka jokin alilohko lavennetaan merkkijonoksi. Tietyn oletusarvoisen alilohkon lavennustavan ohella voidaan rakentaa vaihtoehtoisia lavennustapoja, joita käytetään juuri tietynlaisten alilohkojen laventamiseen. Tämänhetkisessä TIP-toteutuksessa on käytössä kaksi tavallista tapaa tulkita ja laventaa alilohko: erottaa alilohkon synnyttämän merkkijonon alusta tunnus, jota etsitään symbolitaulusta ja toisena tapana ottaa alilohkon arvona sen synnyttämä merkkijono sellaisenaan.

Liite 3. TIP-esimerkkipohjatiedostoja

Kaikissa pohjatiedostoissa esiintyvien tietohakemiston taulujen nimissä on etuliite "tyy_". Olen jättänyt muualla tässä työssä käsitemalleista jne. tuon toteutus-aikaisen etuliitteen pois.

Ensimmäisenä esimerkkipohjana on erittäin yksinkertainen "copyrigh.tip". Se on mukana muistuttamassa, että TIP:llä voidaan tehdä myös pelkkää perinteistä "include"-toimintaa.

```
----- copyrigh.tip -----
*
*    Copyright (C) 1991 Tietosavo Oy. All Rights Reserved.
*
```

Perustyyppien toteutuskuvauksia ei ole välttämätöntä säilöä tietohakemistoon. Seuraava "ingv.tip"-pohja sisältää Ingres:n perustyyppien toteutuksen. Se on tarkoitettu otettavaksi mukaan ("tip_include"), jos tarvitaan laventaa perustyyppien toteutuksia Ingres:lle.

```
----- ingv.tip -----
$(tip_comment
31.03.1991  amn  (viimeisin editoija)

ingv.tip  --  Ingres-perustietotyyppien esittelyt symbolitauluun.

Milloin     Kuka  Mita
31.03.1991  amn   Taulun nimen maksimipituus 24 merkkia.
28.03.1991  amn   "not null with default" kaikkiin kenttiin.
12.03.1991  amn   "ingv_"-etuliite kaikkiin esiteltyihin tunnuksiin.
11.03.1991  amn   Alkuperainen versio.

$)
$(tip_process_only

$(tip_declare_string kaikille not null with default$)

Esitellaan eri perustietotyyppien lavennukset Ingresille ('ingv').
Merkkijono:
$(tip_declare_string ingv_mj_v varchar$)
$(tip_declare_string ingv_mj_k char$)
$(tip_declare_string ingv_mj_ *** Ei v(aihtuva) eika k(iintea) ***$)
$(tip_declare
  $(tip_as_is ingv_tyy_merkkijono                    $) $(ingv_mj_$(1$)$)($(2$))
$(kaikille$)$)

Lukuarvo:
$(tip_declare_string ingv_lu_l float8$)
$(tip_declare_string ingv_lu_k integer4$)
$(tip_declare_string ingv_lu_ *** Ei l(iukuluku) eika k(okonaisluku) ***$)
```

```
$(tip_declare
  $(tip_as_is ingv_tyy_lukuarvo                    $) $(ingv_lu_$(1$)$) $(kaikille

Muut:
$(tip_declare
  $(tip_as_is ingv_tyy_raha                        $)
  money $(kaikille$)$)
$(tip_declare
  $(tip_as_is ingv_tyy_paivamaara                  $)
  char(8) $(kaikille$) /* vvvvkkpp */$)
$(tip_declare
  $(tip_as_is ingv_tyy_aika                        $)
  char(9) $(kaikille$) /* hhmmssddd */$)
$(tip_declare
  $(tip_as_is ingv_tyy_aikaleima                   $)
  char(17) $(kaikille$) /* vvvvkkpphhmmssddd */$)

Sitten TIP-funktio, joka rajaa taulujen tod. nimet:
$(tip_declare
  taulunimirajoitus
  $(tip_str $(1$) 24 Variable Lowercase$)
$)
end-of-process-only$)
```

"ingv.tip":ä vastaava rakenne C- ja CIP-kielisille ohjelmille. Vertaamalla "ingv.tip":iä ja "ccip.tip":iä nähdään, kuinka yhteisestä tietohakemistotiedosta voidaan tuottaa eri välineille erilaisia kuvauksia.

```
----- ccip.tip -----
$(tip_comment
29.03.1991  amn  (viimeisin editoija)

ccip.tip  --  C / CIP -perustietotyyppien esittelyt symbolitauluun.

Milloin     Kuka  Mita
29.03.1991  amn   Tyypillisia CIP-rakenteita.
28.03.1991  amn   "tip_adjust" kayttoon.
27.03.1991  amn   Alkuperainen versio.

$)
$(tip_process_only

Esitellaan eri perustietotyyppien lavennukset C:lle / CIP:lle ('ccip').
Merkkijono:
$(tip_declare_string ccip_mj_v ENFOVARCHAR$)
$(tip_declare_string ccip_mj_k ENFOCHAR   $)
$(tip_declare_string ccip_mj_ *** Ei v(aihtuva) eika k(iintea) ***$)
$(tip_declare
  $(tip_as_is ccip_tyy_merkkijono                  $) $(ccip_mj_$(3$)$)
$(1$)[$(tip_adjust $(4$) +1 $(2$)$)]$)

Lukuarvo:
$(tip_declare_string ccip_lu_l ENFODOUBLE $)
$(tip_declare_string ccip_lu_k ENFOLONG   $)
```

```
$(tip_declare_string ccip_lu_ *** Ei l(iukuluku) eika k(okonaisluku) ***$)
$(tip_declare
  $(tip_as_is ccip_tyy_lukuarvo                    $) $(ccip_lu_$(3$)$) $(1$)$)

Muut:
$(tip_declare
  $(tip_as_is ccip_tyy_raha                        $)
  ENFODOUBLE  $(1$)  /* raha */$)
$(tip_declare
  $(tip_as_is ccip_tyy_paivamaara                  $)
  $($(tip_as_is ccip_tyy_merkkijono                  $) $(1$) $(2$) k 8$)  /* vvvv
*/$)
$(tip_declare
  $(tip_as_is ccip_tyy_aika                        $)
  $($(tip_as_is ccip_tyy_merkkijono                  $) $(1$) $(2$) k 9$)  /*
hhmmssddd */$)
$(tip_declare
  $(tip_as_is ccip_tyy_aikaleima                   $)
  $($(tip_as_is ccip_tyy_merkkijono                  $) $(1$) $(2$) k 17$)  /*
vvvvkkpphhmmssddd */$)


Esitellaan tyypillisia CIP-rakenteita:

Statuksen tarkastus siten, etta lisataan tama metodi virhepinoon.
$(tip_declare_string if_status
   if (status) {
      @status edKdillaMtdi_ mtdi esiint_ self@;
      return (status);
   }
$)

end-of-process-only$)
```

Uudet relaatiotaulut kohteen kenttäluetteloiden avulla antava pohja "ct.tip". Tämä pitää ajaa "kohteet.tip"-pohjan mukaanotettuna pohjatiedostona.

```
----- ct.tip -----
$(tip_comment
22.04.1991  amn  (viimeisin editoija)

ct.tip  --  Taulujen luonti-SQL:n tuottaminen tip:lla

Milloin     Kuka  Mita
22.04.1991  amn   "with journalling" & "grant all".
09.04.1991  amn   Alustavia indeksinluonteja (kommentteihin).
31.03.1991  amn   Taulun nimen (maksimipituuden) rajoittava funktio.
28.03.1991  amn   Include "kohteet.tip".
27.03.1991  amn   Include "kentat.tip".
12.03.1991  amn   Virheilmoitus "vaarat parametrit perustyypille" ("ingv.tip").
11.03.1991  amn   Perustyypin parametrien pilkkominen perustyyppikohtaisten
                  TIP-funktioiden parametreiksi.  Ingv-include.
10.03.1991  amn   Testaillaan seka "tt%"-alkuisilla etta vain "tts_%"-
                  alkuisilla tauluilla.
```

```
08.03.1991  amn   Rajataan taulujen hakua vain "tts_%"-alkuisiin.
06.03.1991  amn   Muistuttamaan vahan ihan oikeaa taulujen luontia.
05.03.1991  amn   Alkuperainen versio.

$)
$(tip_declare_string tiedoston_nimi Ct$(tip_str $(kohteen_lyhyt_tunnus$) 3 Variab:
Capital$)$)
/* $(tip_timestamp$)  $(tip_user$)  (viimeisin TIP) */
/*********************************************************************
*    $(tiedoston_nimi$).isql  --  Taulun "$(kohteen_tunnus$)" luonti-SQL.
*
$(copyright_teksti$)
*********************************************************************/

create table $(taulunimirajoitus $(kohteen_tunnus$)$) (
    $(tip_for_each kentta
      $(  $(lyhyt_tunnus$) $(ingv_$(perustyypin_tunnus$)
                             $(tip_argumentize
                               $(perustyypin_parametrit$)
                             $)
                           $)
      $)
      separator $(,
                $)
    endfor$)

)
with journaling
;

grant all on $(taulunimirajoitus $(kohteen_tunnus$)$) to public;

commit work;

/* TIP paattyi $(tip_timestamp$). */
```

Seuraavassa on esimerkki eräästä tietokannan taulusta, joka on tuotettu pohjalla "ct.tip". Olen poistanut välistä n. 30 kenttää toimialakohtaisesti arkoina.

```
----- ctlas.tip -----
/* 15.07.1991 15:15:53  AMN  (viimeisin TIP) */
/*********************************************************************
*    CtLas.isql  --  Taulun "ttl_lasku" luonti-SQL.
*
*
*    Copyright (C) 1991 Tietosavo Oy. All Rights Reserved.
*
*********************************************************************/

create table ttl_lasku (
  tositelaji                    char(2) not null with default,
  tositenro                     char(8) not null with default,
  laskuttajan_laskunro          varchar(32) not null with default,
  laskupvm                      char(8) not null with default /* vvvvkkpp */,
  valuutta                      char(3) not null with default,
```

```
  kurssi                          float8 not null with default,
  loppusumma_mk                   money not null with default,
  loppusumma_val                  money not null with default,
...
  30 muuta kenttää
...
  erapaiva                        char(8) not null with default /* vvvvkkpp */,
  perustaja                       char(32) not null with default,
  perustettu                      char(17) not null with default /* vvvvkkpphhmms
*/,
  muuttaja                        char(32) not null with default,
  muutettu                        char(17) not null with default /* vvvvkkpphhmms
*/
)
with journaling
;

grant all on ttl_lasku to public;

commit work;

/* TIP paattyi 15.07.1991 15:15:57. */
```

Tietuepuskureiden kuvauksia voidaan tuottaa vaikkapa seuraavanlaisella "koyhth.tip"-pohjalla. Tässä pohjassa näkyy tavallista enemmän mukautumista kohdevälineen vaatimuksiin.

```
----- koyhth.tip -----
$(tip_comment
22.04.1991  amn  (viimeisin editoija)

koyhth.tip  --  Tiedonhallintakohteen tiedot --> "koYhtXxx.h".
                Yhteiset kuvaukset kohteesta tiedonhallinnan luokkien
                "Ko/Ta/RiThaXxx.pre" ja sovelluksen kohdeluokan
                "KoSovXxx.pre" kayttoon.

Milloin     Kuka  Mita
22.04.1991  amn   Hakuehtoja lisaa.
09.04.1991  amn   Hakuehdot alustavasti mukaan.
08.04.1991  amn   Nakyma pois, hakuehdot kommentteihin.
28.03.1991  amn   "tip_adjust" kayttoon.
27.03.1991  amn   Alkuperainen versio.
$)
$(tip_declare_string tiedoston_nimi KoYht$(tip_str $(kohteen_lyhyt_tunnus$) 3 Var
Capital$)$)
/* $(tip_timestamp$)  $(tip_user$)  (viimeisin TIP) */
/*********************************************************************
*
*    Tdston nimi:  $(tiedoston_nimi$).h
*                  -- Kohteen "$(kohteen_tunnus$)"
*                     struct-kuvaukset, kenttien sanomanimet yms.
*
*    Kuvaus:       Tiedonhallinnan luokkien
("Ko/Ta/RiTha$(kohteen_lyhyt_tunnus$).pre")
```

```
*                   ja sovelluksen kohdeluokan ("KoSov$(kohteen_lyhyt_tunnus$).pre'
*                   valilla yhteiset maarittelyt:
*                    - sanomissa kulkeva struct-typedef ST$(kohteen_lyhyt_tunnus$)
vastaava osoitin STPTR$(kohteen_lyhyt_tunnus$)

*                    - struct-typedef DB$(kohteen_lyhyt_tunnus$) ja
*                      vastaava osoitin DBPTR$(kohteen_lyhyt_tunnus$),
*                      joiden sisaltamat kentat ovat juuri oikean pituisia
*                      eivatka 4:lla jaolliseen tavukokoon pyoristettyja
*                    - #define-symbolit kenttien lukumaaralle ja
*                      jokaista kenttaa kohti:
*                       - symboli kentan jarjestysnumerolle
*                       - symboli kentan sanomanimelle
*                    - #define-symbolit taman kohteen hakuehtojen sanomanimille
*                      ja itse kohteen structin sanomanimelle / sanomaotsikon
*                      arvolle
*
*    Tekija:        Ari Mujunen 27.03.1991,
*                   pohjana Tuomo Vehosmaan 9.10.1990 alkaen tekema
*                   sijoitusten seurannan "koyhtsij.h".
$(copyright_teksti$)
************************************************************************/


/*
 * Tietuekuvaukset (structit).
 */

/* "ENFO_THASQLLK" on mukana, jos tiedosto menee ESQL-esikaantajan lapi. */
#ifdef ENFO_THASQLLK
exec sql begin declare section;
#endif

/*
 * Ensin struct, jota kaytetaan struct-muotoisten kohteiden kuljettamiseen
 * sanomissa.  Jokaisen kentan pituus on pyoristetty ylospain neljalla
 * jaolliseen tavukokoon (riippumattomuus erimerkkisista C-kaantajista).
 */
typedef struct {
    $(tip_for_each kentta
      $(  $(ccip_$(perustyypin_tunnus$)
              $(tip_str $(lyhyt_tunnus$) 32 Variable$)
              4
              $(tip_argumentize
                  $(perustyypin_parametrit$)
              $)
          $);
      $)
    endfor$)
} ST$(kohteen_lyhyt_tunnus$), *STPTR$(kohteen_lyhyt_tunnus$);

#ifdef ENFO_THASQLLK
exec sql end declare section;
#endif
```

```
/*
 * Tata structia kaytetaan toistaiseksi vain kenttien todellisten pituuksien
 * valittamiseen kaantajalle.  Luokissa viitataan "sizeof(p->kentta)"-
 * rakenteella taman structin kenttiin ja saadaan nain kenttien todelliset
 * pituudet (eli tietokantapituudet; tasta lyhenne "DB") kayttoon.
 */
typedef struct {
    $(tip_for_each kentta
      $(  $(ccip_$(perustyypin_tunnus$)
              $(tip_str $(lyhyt_tunnus$) 32 Variable$)
              1
              $(tip_argumentize
                  $(perustyypin_parametrit$)
              $)
          $);
      $)
    endfor$)
} DB$(kohteen_lyhyt_tunnus$), *DBPTR$(kohteen_lyhyt_tunnus$);


/*
 * Kohteen "$(kohteen_tunnus$)" kenttien taulukkoindeksit.
 */
#define ST$(kohteen_lyhyt_tunnus$)KENTTALKM $(kentta.tip_rowcount$)

$(tip_for_each kentta
  $(
#define ST_$(kentan_tunnus_defineen$)  $(tip_adjust $(kentta.i$) -1$)

  $)
endfor$)


/*
 * Olion nimi (sanomassa "YL_OTSIKKO_ALKU/LOPPU"-avaimen arvona
 * seka structin avaimena).
 */
#define $(tip_str $(osajarjestelma$)_S_$(kohteen_lyhyt_tunnus$)_OLIO 80 None
Uppercase$)  "$(tip_str $(kohteen_tunnus$)_OLIO 32 Fixed Uppercase$)"

/*
 * Hakuehdot.  ??? Alustava yrite:
 */
$(tip_for_each hakuehto
  $(
#define $(hakuehdon_tunnus_defineen$)  "$(tip_str
$(kohteen_lyhyt_tunnus$)_$(hakuehdon_tunnus$) 32 Fixed Uppercase$)"
  $)
endfor$)

/*
 * Kohteen "$(kohteen_tunnus$)" kenttien sanomanimet.
 */
$(tip_for_each kentta
  $(
```

```
#define $(kentan_tunnus_defineen$)  "$(tip_str
$(kohteen_lyhyt_tunnus$)_$(lyhyt_tunnus$) 32 Fixed Uppercase$)"
  $)
endfor$)

/* $(tip_timestamp$)  $(tip_user$)  (TIP paattyi) */
```

Edellä esitellyt "ct.tip" ja "koyhth.tip" eivät toimi itsenäisinä, vaan vain osana "kohteet.tip":ä. Tällä tavalla pohjia saatiin lyhennettyä.

```
----- kohteet.tip -----
$(tip_comment
15.07.1991  amn  (viimeisin editoija)

kohteet.tip  --  Valituille kohteille tehdaan kohteeseen liittyvat pohjat.

Milloin     Kuka  Mita
15.07.1991  amn   Haluttujen kohteiden valintaehto ja
                  haluttujen pohjien valintavivut erilliseen tiedostoon '$(1$)'.
                  Samalla yhteinen etuliite muuttui nimelle '$(2$)'.
11.07.1991  pjt   Toimintoluokka mukaan
22.04.1991  amn   Yhteinen etuliite "$(1$)" kaikille tuotetuille tiedostoille.
09.04.1991  amn   Kaikki kohteen tarv. tiedostot kerralla.
28.03.1991  amn   Alkuperainen versio.

$)
$(tip_comment
---
Komentorivin argumenttien talletus symbolitauluun selvakielisille nimille.
$)
$(tip_declare_string kohteiden_ja_pohjien_valintatiedoston_nimi $(1$)$)
$(tip_declare_string tulostiedostojen_hakemisto $(2$)$)
$(tip_comment
---
Aina voimassa olevia maarittelyja.
$)
$(tip_declare_string jarjestelma ENFO$)
$(tip_declare_string osajarjestelma EFEKTO$)
$(tip_include ingv.tip$)
$(tip_include ccip.tip$)
$(tip_declare_string_by_including copyright_teksti copyrigh.tip$)
$(tip_declare_by_including kentat kentat.tip$)
$(tip_declare_by_including hakuehdot hakuehdo.tip$)
$(tip_comment
----- Kohteeseen liittyvat pohjat:
Esitellaan ensin oletusarvot: kasitellaan pari sijoituskohdetta ja kaikki pohjat
$)
$(tip_declare_string kohteiden_valintaehto where tunnus like 'tts_sij%'$)
$(tip_declare_string ct_vipu k$)
$(tip_declare_string koyhth_vipu k$)
$(tip_comment
-----
Haetaan kayttajan valitsemasta tiedostosta oletusarvojen paalle
haluttu pohjaehto ja haluttujen pohjien vipuset.
$)
```

```
$(tip_include $(kohteiden_ja_pohjien_valintatiedoston_nimi$)$)
$(tip_comment
---
lue_pohja_vain_jos_tarvitaan:
1 = Pohjan nimiosa
$)
$(tip_declare lue_pohja_vain_jos_tarvitaan
  $(tip_case $($(1$)_vipu$)
    when k $(
      $(tip_declare_by_including $(1$)pohja $(1$).tip$)
    $)
  endcase$)
$)
$(lue_pohja_vain_jos_tarvitaan ct$)
$(lue_pohja_vain_jos_tarvitaan koyhth$)
$(tip_sql kohde
  select
    tunnus,
    lyhyt_tunnus
  from
    tyy_kohde
    $(kohteiden_valintaehto$)
  order by
    tunnus
$)
$(tip_comment
---
1 = Pohjan symbolitaulutunnukset etuliitteenomainen nimi xxpohja.
2 = Tulostiedoston vakio-osa (jonka peraan tulee kolmikirjaiminen kohdelyhenne).
3 = Lisa-argumentti pohjalle.
4 = Tiedoston loppuliite.
$)
$(tip_declare pursota_yksi_pohja
  $(tip_case $($(1$)_vipu$)
    when k $(
      $(tip_change_output_file
$(tulostiedostojen_hakemisto$)$(2$)$(kohteen_lyhyt_tunnus$)$(4$)$)
      $($(1$)pohja $(3$)$)
    $)
  endcase$)
$)
$(tip_declare_string tyhja$)
$(tip_for_each kohde
  $(
    $(tip_declare_string kohteen_tunnus $(tip_str $(tunnus$) 32 Variable$)$)
    $(tip_declare_string kohteen_lyhyt_tunnus $(tip_str $(lyhyt_tunnus$) 3 Variab
Uppercase$)$)
    $(kentat $(kohteen_tunnus$)$)
    $(hakuehdot $(kohteen_tunnus$)$)
    $(pursota_yksi_pohja ct Ct $(tyhja$) .isql$)
    $(pursota_yksi_pohja koyhth KoYht $(tyhja$) .h$)
    $)
endfor$)
```

Kohteiden kenttien "litistys" eli haku tietohakemiston tyyppijärjestelmästä tehdään Esitetyssä "tihlit.tip"-versiossa ei ole otettu huomioon tietohakemiston tyyppien toteutuskuvausta eikä versiotunnuksen käyttöä kenttäluetteloa talletettaessa. Mukana ei ole tiedosto kokonaisuudessaan, vaan siihen asti, kun sisäkkäiset tyyppiviittaukset purkava TIP-aliohjelma on määritelty.

```
----- tihlit.tip -----
$(tip_comment
31.07.1991  amn  (viimeisin editoija)

tihlit.tip  --  Valittujen kohteiden kenttien tekeminen tyyppien perusteella
                eli "litistys".

Milloin     Kuka  Mita
31.07.1991  amn   Kohteiden valintaehdon haku erilliseen includeen 'tihkohva.tip'.
16.07.1991  amn   Toimimaan halutun taulun sisalla jatkuu.
15.07.1991  amn   Toimimaan halutun taulun sisalla (so. ei kahdessa taulussa).
12.07.1991  amn   Paivitys haluttuun tauluun.
11.07.1991  amn   Alkuperainen versio.

$)
-----------------------------------------
Kohteiden kenttien paivitys eli "litistys".
$(tip_user$)  $(tip_timestamp$)

-----------------------------------------

$(tip_declare_string tih_etuliite tyy$)
$(tip_declare_string ei_kaytossa_oleva_rivinumero '****'$)
$(tip_comment
---
Litistys tapahtuu tauluun 'uudet_kentat'.
Lyhyet tunnukset kopioidaan taulusta 'vanhat_kentat'.
Nama kaksi taulua voivat olla sama taulu.$)
$(tip_declare_string vanhat_kentat $(tih_etuliite$)_kohteen_kentta$)
$(tip_declare_string uudet_kentat $(tih_etuliite$)_kohteen_kentta$)
$(tip_comment
---
Haetaan 'kohteet.tip':n kanssa yhteinen include, joka maarittelee
tunnuksen 'kohteiden_valintaehto', jolla valitaan ("where"-ehto) halutut kohteet
tai vaihtoehtoisesti otetaan ehto suoraan komentorivilta.$)
$(tip_include tihkohva.tip $(1$) $(2$)$)
$(tip_declare_string kohteiden_aliselect_valintaehto
  where kohteen_tunnus in (
    select tunnus from $(tih_etuliite$)_kohde
    $(kohteiden_valintaehto$)
  )
$)
$(tip_comment
---$)
$(tip_declare luo_taulu
  $(tip_excel_sql
    create table $(uudet_kentat$) (
```

```
      kohteen_tunnus char(32) not null not default,
      nro char(4) not null not default,
      lyhyt_tunnus char(32) not null with default,
      tunnus vchar(255) not null not default,
      perustyypin_tunnus char(32) not null not default,
      perustyypin_parametrit vchar(255) not null with default
    )
  $)
$)
$(tip_comment
---
Tutkitaan, pitaisiko uusien kenttien taulu luoda.
Mikali se on olemassa, poistetaan siita nyt litistettavien taulujen kentat.$)
$(tip_sql riveja_taulussa
  select
    riveja = count(*)
  from
    $(uudet_kentat$)
$)
$(tip_case $(riveja_taulussa.tip_ok$)
  when True $(
    $(tip_case $(riveja_taulussa.1.riveja$)
      when 0 $(
Uusien kenttien taulu '$(uudet_kentat$)' on tyhja.
      $)
      otherwise $(
        $(tip_case $(vanhat_kentat$)
          when $(uudet_kentat$) $(
Merkitaan vanhat kentat tauluun '$(vanhat_kentat$)'...
            $(tip_excel_sql
               update $(vanhat_kentat$)
               set
                 nro = $(ei_kaytossa_oleva_rivinumero$)
$(kohteiden_aliselect_valintaehto$)
            $)
          $)
          otherwise $(
Poistetaan vanhat kentat taulusta '$(uudet_kentat$)'...
            $(tip_excel_sql
              delete from $(uudet_kentat$) $(kohteiden_aliselect_valintaehto$)
            $)
          $)
        endcase$)
      $)
    endcase$)
  $)
  otherwise $(
Taulua '$(uudet_kentat$)' ei ole ennestaan olemassa.
Luodaan taulu '$(uudet_kentat$)'...
    $(luo_taulu$)
  $)
endcase$)
$(tip_excel_sql commit work$)
$(tip_comment
---
```

```
Esitellaan sisakkain olevat ryhma-, yksinkertaiset ja perustyypit purkavat
TIP-aliohjelmat.
'pura_tyyppi'
  1 = tyypin tunnus
  2 = tahan mennessa keratty etuliite
  3 = mahdolliset perustyypin parametrit.$)
$(tip_declare pura_tyyppi
  $(tip_sql_once $(1$)_tyyppi
    select
      laji
    from
      $(tih_etuliite$)_tyyppi
    where
      tunnus = '$(1$)'
  $)
  $(tip_case $($(1$)_tyyppi.tip_rowcount$)
    when 1 $(
      $(pura_tyyppi_$($(1$)_tyyppi.1.laji$) $(1$) $(2$) $(3$)$)
    $)
    when 0 $(
      $(pura_tyyppi_ $(1$) $(2$) $(3$)$)
    $)
    otherwise $(
      *** Tyypilla '$(1$)' on useita maarityksia ($($(1$)_tyyppi.tip_rowcount$)).
      $(pura_tyyppi_$($(1$)_tyyppi.1.laji$) $(1$) $(2$) $(3$)$)
    $)
  endcase$)
$)
$(tip_comment
---
Tyyppikohtaiset purkualiohjelmat.
'pura_tyyppi_'   Tuntematon
'pura_tyyppi_P'  Perustyyppi
'pura_tyyppi_Y'  Yksinkertainen tyyppi
'pura_tyyppi_R'  Ryhmètyyppi.
Parametrit ovat kaikilla samoin kuin 'pura_tyyppi':lla eli:
  1 = tyypin tunnus
  2 = tahan mennessa keratty etuliite
  3 = mahdolliset perustyypin parametrit.$)
$(tip_declare pura_tyyppi_
$(2$): *** Tyyppi '$(1$)' on tuntematon (ei ole P/Y/R).
$)
$(tip_declare pura_tyyppi_P
  $(kentan_numero$) $(2$): $(tip_str $(1$) 32 Variable$)($(3$));
  $(tip_message .$)
  $(tip_excel_sql
    insert into $(uudet_kentat$) (
      kohteen_tunnus,
      nro,
      tunnus,
      perustyypin_tunnus,
      perustyypin_parametrit
    ) values (
      '$(kohteen_tunnus$)',
      '$(kentan_numero$)',
      '$(2$)',
```

```
      '$(1$)',
      '$(3$)'
    )
  $)
  $(tip_declare_string kentan_numero $(tip_adjust $(kentan_numero$) +10 1 %04ld$)$
$)
$(tip_declare pura_tyyppi_Y
  $(tip_sql_once $(1$)_yks
    select
      tyypin_tunnus = ifnull(alityypin_tunnus, perustyypin_tunnus),
      perustyypin_parametrit
    from
      $(tih_etuliite$)_yksinkertainen_tyypp t
    where
      t.tyypin_tunnus = '$(1$)'
  $)
  $(pura_tyyppi
    $($(1$)_yks.1.tyypin_tunnus$) $(2$) $($(1$)_yks.1.perustyypin_parametrit$)
  $)
$)
$(tip_declare pura_tyyppi_R
  $(tip_sql_once $(1$)_kentta
    select
      nro,
      tunnus = trim(tunnus),
      tyypin_tunnus = ifnull(alityypin_tunnus, perustyypin_tunnus),
      perustyypin_parametrit
    from
      $(tih_etuliite$)_ryhmatyypin_kentta
    where
      ryhmatyypin_tunnus = '$(1$)'
    order by
      nro
  $)
  $(tip_for_each $(1$)_kentta $(
    $(pura_tyyppi
      $(tyypin_tunnus$) $(2$).$(tunnus$) $(perustyypin_parametrit$)
    $)
  $)
  endfor$)
$)
$(tip_comment
---
Maaritellaan viela tyhja merkkijono, jota on muuten vaikea saada parametriksi.$)
$(tip_declare_string tyhja$)
$(tip_comment
---
Nyt tyypinpurkualiohjelmat ovat valmiita kayttoon.
Haetaan halutut kohteet ja puretaan kunkin kohteen kuvaava ryhmatyyppi.$)
...
```